2004:228 CIV

EXAMENSARBETE

Convergence Time Redution in the BGP4 Routing Protocol Using the "Ghost-Flushing" Technique and Other Proposals

Gonzalo P. Rodrigo Álvarez

MASTER OF SCIENCE PROGRAMME

Department of Computer Science and Electrical Engineering
Division of Computer Science

LULEÅ TEKNISKA UNIVERSITET

2004:228 CIV • ISSN: 1402 - 1617 • ISRN: LTU - EX - - 04/228 - - SE

Convergence Time Reduction in the BGP4 Routing Protocol Using the "Ghost-Flushing" Technique and Other Proposals

Gonzalo P. Rodrigo Álvarez

Luleå, July, 2004

Soy un ciudadano del mundo entero.

I am a citizen of the whole world.

Erasmo de Rotterdam, humanista holandés del siglo XV.
Erasmo of Rotterdam, Dutch humanist from the 15^{th} *Century.*

Abstract

BGP4 (Border Gateway Protocol) is the language that makes possible to keep up to date the road maps in the internet. This routing protocol is the one used by the "big networks" to exchange routing information. Thus, its performance is critical for the internet.

In some situations the behavior of this protocol is not the desired one. This thesis (Or project) intends to analyze one of this problems together with some proposed solutions. This analysis will involve a review of the theory behind the proposals and a later simulation to test the performance of them. After the analysis is done, the proposals considered as "useful" will be implemented over an open source, free routing software, GNU Zebra.

This work began as the analysis of the main proposal, the Ghost-Flushing rule.

BGP4 (Border Gateway Protocol) es el idioma que hace posible que los mapas de carretera de internet estén siempre actualizados. Este protocolo es utilizado por las "redes corporativas" para intercambiar información de enrutado, por lo tanto, su comportamiento es crítico para el buen funcionamiento de Internet.

En algunos casos, el comportamiento de este protocolo no es el deseado. Este proyecto pretende analizar uno de estos problemas junto con una serie de propuestas para solucionarlo. Todo esto implicará revisar la teoría que sustenta estas propuestas y, más tarde, realizar una serie de simulaciones para evaluar su rendimiento. Después, aquellas propuestas que podamos considerar como útiles serán implementadas sobre un software de enrutado gratuito y de código abierto, GNU Zebra.

Todo este proyecto comenzó como el análisis de la principal de todas las propuestas, la regla Ghost-Flushing.

Contents

I Complete Report **xix**

1 Introduction **1**
1.1 Motivation, Purpose and Goal . . . 1
1.2 Structure and method of this work . . . 1
1.3 Chapters Overview . . . 2

2 BGP4 Routing Protocol and its Environment **3**
2.1 Network Routing . . . 3
2.1.1 Hop-by-hop Routing . . . 3
2.1.2 Mapping the Internet . . . 3
2.1.3 Exterior/Interior routing . . . 4
2.2 BGP4 Protocol Description . . . 5
2.2.1 Path Vectors and Loop Detection . . . 5
2.2.2 Path Attributes . . . 6
2.2.3 The Protocol . . . 6
2.2.4 CIDR support and BGP4 . . . 8
2.2.5 Internal BGP (IBGP) . . . 9
2.3 Autonomous System Relationships . . . 10
2.3.1 Relationship Classification and Internet Disposition . . . 11
2.4 History . . . 11

3 Ghost Flushing and other Proposals **13**
3.1 The Convergence Problem . . . 13
3.1.1 Some initial concepts . . . 13
3.1.2 High Convergence Time and The Ghost Information . . . 13
3.2 Ghost Flushing Rule . . . 15
3.2.1 How it works . . . 15
3.2.2 Convergence time . . . 16
3.3 Ghost Buster Rule . . . 18
3.3.1 How it works . . . 18
3.3.2 Convergence time . . . 19
3.4 Other Solutions . . . 19
3.4.1 Reset Rules . . . 19
3.4.2 Other Approaches to the same problem . . . 19

4 BGP4 Simulation Environment **21**
4.1 Choosing a Simulator . . . 21
4.1.1 NS-2 and BGP++ . . . 21
4.1.2 SSFNet 2.0 . . . 22
4.2 Simulation Environment . . . 24
4.2.1 Simulation Parameters and Method . . . 24
4.2.2 Simulating means programming . . . 25

4.2.3 The Hardware . . . 31
4.3 Analysis of SSFNet's BGP4 Implementation . . . 32
4.3.1 Source Structure . . . 32
4.3.2 BGPSession Class . . . 33
4.3.3 The Update Process . . . 36
4.4 Modifications to SSFNet's BGP4 for the Ghost-Flushing Rule . . . 37
4.4.1 Activating The Ghost-Flushing feature from the DML file . . . 37
4.4.2 Modification Schema . . . 37
4.4.3 Deeper Inside the Modification (Pseudocode) . . . 37
4.5 Modifications to SSFNet's BGP4 for the Ghost-Buster Rule . . . 41
4.5.1 Activating The Ghost-Buster feature from the DML file . . . 41
4.5.2 The main idea for the modification . . . 41
4.5.3 Making this complex schema work . . . 41
4.5.4 Deep inside the code (Pseudocode) . . . 42
4.5.5 Data structures needed to make all this work . . . 44
4.5.6 Class Schema . . . 45

5 BGP4 Cases Simulation 47
5.1 Simulated Cases . . . 47
5.1.1 Clique . . . 47
5.1.2 Alternate Path (AltPath) . . . 47
5.1.3 "White" model . . . 48
5.1.4 "Bad" model . . . 48
5.1.5 "Multi" Collection . . . 49
5.2 Simulation Results, Split-horizon disabled, No jitter . . . 50
5.2.1 Clique Topology . . . 51
5.2.2 AltPath Topology . . . 52
5.2.3 Multi Topology . . . 54
5.2.4 White Topology . . . 55
5.2.5 Bad Case . . . 57
5.3 Simulation Results, Split-horizon disabled, Jitter Activated . . . 58
5.3.1 Clique Topology . . . 58
5.3.2 AltPath Topology . . . 60
5.3.3 Multi Topology . . . 61
5.3.4 White Topology . . . 62
5.3.5 Bad Case . . . 63
5.3.6 Comparing between jitter enabled and disabled . . . 64
5.4 Simulation Results, Split-horizon Enabled, No Jitter . . . 65
5.4.1 Clique Topology . . . 65
5.4.2 AltPath Topology . . . 67
5.4.3 Multi Topology . . . 68
5.4.4 Comparing between having or not Split-Horizon . . . 69
5.5 Simulation Results, Split-horizon Enabled, Jitter Activated . . . 70
5.5.1 Clique Topology . . . 70
5.5.2 AltPath Topology . . . 71
5.5.3 Multi Topology . . . 73
5.6 Comparing all the techniques . . . 74
5.7 Combining nodes with and without ghost-flushing . . . 74
5.8 Conclusions . . . 75
5.8.1 Ghost-Flushing . . . 75
5.8.2 Ghost-Buster . . . 76

6 Applying the Ghost Flushing Rule to Zebra Routing Software 77
6.1 Zebra as a Routing Software . . . 77
6.1.1 History . . . 77
6.1.2 Features . . . 78
6.1.3 Setting Up a Zebra/BGP speaker . . . 78
6.2 Architecture of Zebra Routing Software . . . 79
6.2.1 Modules, Daemons and Sockets . . . 79
6.2.2 Zebra Code Structure . . . 80
6.3 Architecture of bgpd . . . 81
6.3.1 A Thread's life . . . 81
6.3.2 Good Daemons speak Zebra Protocol . . . 82
6.3.3 bgpd source files . . . 82
6.3.4 A bgpd Daemon is born . . . 83
6.3.5 From an Update Arrival to an Update Departure . . . 83
6.3.6 The VTY interface . . . 85
6.4 Modifications to the code of bgpd . . . 86
6.4.1 Implementing ghost-flushing over GNU Zebra . . . 86
6.4.2 Modifying the VTY configuration environment . . . 86
6.5 Tests . . . 89
6.5.1 Testing and Developing Environment . . . 89
6.5.2 Testing means Programming . . . 91
6.5.3 Tests and results . . . 91
6.6 Conclusions . . . 92
6.6.1 General Conclusions . . . 92
6.6.2 Odd things about GNU Zebra . . . 92

7 Conclusions and Discussions 93
7.1 Quality of the Simulation Results . . . 93
7.2 The Ghost Flushing Rule . . . 93
7.3 The GNU Zebra Routing Software . . . 94

8 Future Work 95
8.1 Interaction of Ghost-Flushing with other features . . . 95
8.2 Testing Zebra to its limits . . . 95

II Resumen en Español 97

1 Introducción 99
1.1 Motivación, propósito y objetivos . . . 99
1.2 Estructura y método de este trabajo . . . 100

2 Protocolo de encaminamiento BGP4 y su entorno 101
2.1 Descripción del Protocolo BGP4 Protocol . . . 101
2.1.1 Vectores de Ruta y Detección de ciclos . . . 101
2.1.2 Atributos de ruta . . . 101
2.1.3 El Protocolo . . . 102
2.1.4 Soporte a CIDR . . . 104
2.1.5 BGP Interno (IBGP) . . . 105
2.2 Relaciones entre Sistemas Autónomos . . . 106
2.2.1 Clasificación de Relaciones y Disposición en Internet . . . 107

3 Ghost Flushing y Otras Propuestas **109**
3.1 El Problema de la Convergencia . . . 109
3.1.1 Algunas Ideas Iniciales . . . 109
3.1.2 Tiempo de Convergencia Elevado y la Información Fantasma . . . 110
3.2 Regla Ghost Flushing . . . 112
3.2.1 Como Funciona . . . 112
3.2.2 Tiempo de convergencia . . . 113
3.3 Regla Ghost-Buster . . . 114
3.3.1 Como Funciona . . . 114
3.3.2 Tiempo de Convergencia . . . 114
3.4 Otras Soluciones . . . 115
3.4.1 Reglas de Reset . . . 115
3.4.2 Otros enfoques sobre el mismo problema . . . 115

4 Entorno de Simulación BGP4 **117**
4.1 Simulador y casos Simulados . . . 117
4.2 Modificaciones a la implementación para la regla Ghost-Flushing . . . 117
4.2.1 Activando la regla Ghost-Flushing desde el archivo DML . . . 117
4.2.2 Esquema de modificación Modification . . . 118
4.3 Modificaciones al la Implementación regla Ghost-Buster . . . 118
4.3.1 Activando la regla Ghost-Buster desde el archivo DML . . . 118
4.3.2 La idea detrás de la modificación . . . 119
4.3.3 Como funciona todo este lío . . . 119

5 Simulación de Casos de BGP4 **121**
5.1 Comparando todas las técnicas . . . 121
5.2 Combinando nodos con y sin ghost-flushing . . . 121
5.3 Conclusiones . . . 122
5.3.1 Ghost-Flushing . . . 122
5.3.2 Ghost-Buster . . . 123

6 Aplicando la regla Ghost-Flushing a GNU Zebra **125**
6.1 Zebra como Software de Encaminado . . . 125
6.1.1 Funcionalidades . . . 125
6.2 Arquitectura de GNU Zebra . . . 126
6.2.1 Módulos, Demonios y Sockets . . . 126
6.3 Tests . . . 127
6.3.1 Tests y resultados . . . 127
6.4 Conclusiones . . . 127
6.4.1 Conclusiones Generales . . . 127

7 Conclusiones Y Discusiones **129**
7.1 Calidad de los resultados de Simulación . . . 129
7.2 La regla Ghost-Flushing . . . 129
7.3 Software de Encaminado GNU Zebra . . . 129

8 Trabajo Futuro **131**
8.1 Interacción de Ghost-Flushing con otras reglas . . . 131
8.2 Probando Zebra hasta su Límites . . . 131

III Appendix **133**

A BGP with Ghost Flushing Pseudocode **135**

B Zebra bgpd node configuration file example **137**

C DML 5 nodes clique scenario **139**

D SSFNet Simulation execution Makefile **143**

E "Multi" Network Models **145**

F Zebra Testing Lab **147**

G GNU Zebra log file example **149**

H Digital Media Contents **151**

List of Figures

2.1 Example of hop-by-hop routing between two hosts . . . 3
2.2 Routers communicating reachabillity information. . . . 4
2.3 New picture of the AS-divided Internet. . . . 5
2.4 Schema (Simplified) of the Update handling by SSFNet BGP4 Implementation. . . . 8
2.5 IP classes distribution . . . 9
2.6 Example of a Route Reflector Configuration inside the same AS. . . . 10
2.7 Filter Policies. . . . 10
2.8 Chaotic example of Internet providers interconnection. . . . 11

3.1 Example of Ghost Information spreading in a clique topology and high E_{down}. . . . 14
3.2 Example of Ghost Information spreading in a fail-over scenario. . . . 15
3.3 Ghost Flushing Rule. . . . 16
3.4 Step by step, Ghost Flushing Rule in action. . . . 16
3.5 Bad situation for ghost-flushing. . . . 18
3.6 Ghost Buster rule. . . . 18

4.1 SSFNet logo. . . . 22
4.2 Two extremes between reality and theory. . . . 25
4.3 Simulation schema. . . . 27
4.4 Interaction between Sigmas and a local account. . . . 32
4.5 LTU's Dator Hall, the Sigmas are the purple machines in the middle-top. Photo by courtesy of Johan Hallbäck. . . . 32
4.6 SSFNet's BGP4 code structure. . . . 33
4.7 Update Process in SSFNet's BGP Schema. . . . 36
4.8 Ghost Flushed Update Process in SSFNet's BGP Schema. . . . 38
4.9 Route arrives and the protocol tries to send if but delta timer is not expired. . . . 42
4.10 Delta Timer Expires. . . . 43
4.11 Ghost Buster modification, Classes interaction. . . . 45

5.1 Full connected 5 nodes clique. . . . 47
5.2 Alternate Path scenario of 11 nodes. The alternate path has 5 nodes ($\frac{n}{2}$) and the clique 6 ($n - \frac{n}{2}$). . . . 48
5.3 "White topology". . . . 48
5.4 "Bad topology'. . . . 49
5.5 "29 nodes structure". Image Generated by RaceWay network visualization tool . . . 50
5.6 Clique Set Up time, no Jitter, No Split-horizon. . . . 51
5.7 Clique Set Up Messages, no Jitter, No Split-horizon. . . . 51
5.8 Clique Fail Down time, no Jitter, No Split-horizon. . . . 51
5.9 Clique Fail Down messages, no Jitter, No Split-horizon. . . . 52
5.10 AltPath Set Up time, no Jitter, No Split-horizon. . . . 52
5.11 AltPath Set Up Messages, no Jitter, No Split-horizon. . . . 53
5.12 AltPath Fail Over time, no Jitter, No Split-horizon. . . . 53
5.13 AltPath Fail Over messages, no Jitter, No Split-horizon. . . . 53
5.14 Multi Set Up time, no Jitter, No Split-horizon. . . . 54

5.15 Multi Set Up Messages, no Jitter, No Split-horizon. . . . 54
5.16 Multi Fail time, no Jitter, No Split-horizon. . . . 54
5.17 Multi Fail messages, no Jitter, No Split-horizon. . . . 55
5.18 White Fail down time, no Jitter, No Split-horizon. . . . 55
5.19 White Fail down time, no Jitter, No Split-horizon. . . . 56
5.20 White Fail down time, no Jitter, No Split-horizon. . . . 56
5.21 White Fail down time, no Jitter, No Split-horizon. . . . 56
5.22 White Fail time down, no Jitter, No Split-horizon. . . . 57
5.23 Bad Fail over time, no Jitter, No Split-horizon. . . . 57
5.24 Bad Fail over time, no Jitter, No Split-horizon. . . . 58
5.25 Clique Set Up time, Jitter, No Split-horizon. . . . 58
5.26 Clique Set Up Messages, no Jitter, No Split-horizon. . . . 59
5.27 Clique Fail Down time, Jitter, No Split-horizon. . . . 59
5.28 Clique Fail Down messages, Jitter, No Split-horizon. . . . 59
5.29 AltPath Set Up time, Jitter, No Split-horizon. . . . 60
5.30 AltPath Set Up Messages, no Jitter, No Split-horizon. . . . 60
5.31 AltPath Fail Over time, Jitter, No Split-horizon. . . . 60
5.32 AltPath Fail Over messages, Jitter, No Split-horizon. . . . 61
5.33 Multi Set Up time, Jitter, No Split-horizon. . . . 61
5.34 Multi Set Up Messages, Jitter, No Split-horizon. . . . 61
5.35 Multi Fail time, Jitter, No Split-horizon. . . . 62
5.36 Multi Fail messages, Jitter, No Split-horizon. . . . 62
5.37 White Fail down time, Jitter, No Split-horizon. . . . 62
5.38 White Fail down time, Jitter, No Split-horizon. . . . 63
5.39 Bad Fail over time, Jitter, No Split-horizon. . . . 63
5.40 Bad Fail over time, Jitter, No Split-horizon. . . . 64
5.41 Comparing the advantage of using jitter on timers. . . . 64
5.42 Clique Set Up time, No Jitter,Split-horizon. . . . 65
5.43 Clique Set Up Messages, No Jitter,Split-horizon. . . . 65
5.44 Clique Fail Down time, No Jitter,Split-horizon. . . . 65
5.45 Clique Fail Down messages, No Jitter,Split-horizon. . . . 66
5.46 AltPath Set Up time, No Jitter, Split-horizon. . . . 67
5.47 AltPath Set Up Messages, No Jitter, Split-horizon. . . . 67
5.48 AltPath Fail Over time, No Jitter, Split-horizon. . . . 67
5.49 AltPath Fail Over messages, No Jitter, Split-horizon. . . . 68
5.50 Multi Set Up time, no Jitter, Split-horizon. . . . 68
5.51 Multi Set Up Messages, no Jitter, Split-horizon. . . . 68
5.52 Multi Fail time, no Jitter, Split-horizon. . . . 69
5.53 Multi Fail messages, no Jitter, Split-horizon. . . . 69
5.54 Comparing between having or not Split-Horizon. . . . 70
5.55 Clique Set Up time, Jitter, Split-horizon. . . . 70
5.56 Clique Set Up Messages, no Jitter, Split-horizon. . . . 70
5.57 Clique Fail Down time, Jitter, Split-horizon. . . . 71
5.58 Clique Fail Down messages, Jitter, Split-horizon. . . . 71
5.59 AltPath Set Up time, Jitter, Split-horizon. . . . 71
5.60 AltPath Set Up Messages, no Jitter, Split-horizon. . . . 72
5.61 AltPath Fail Over time, Jitter, Split-horizon. . . . 72
5.62 AltPath Fail Over messages, Jitter, Split-horizon. . . . 72
5.63 Multi Set Up time, Jitter, Split-horizon. . . . 73
5.64 Multi Set Up Messages, Jitter, Split-horizon. . . . 73
5.65 Multi Fail time, Jitter, Split-horizon. . . . 73
5.66 Multi Fail messages, Jitter, Split-horizon. . . . 74
5.67 Multi E_{down}. . . . 74
5.68 Mixing BGP4 with ghost-flushing. More is worse. . . . 75

6.1 Zebra modular structure. All routing information has to pass through the Zebra daemon. 79

6.2 Here we find the directory structure for the code of Zebra. . . . 80
6.3 Thread life-cycle inside of any of the Zebra Daemons. . . . 81
6.4 bgpd daemon initialization process. . . . 83
6.5 Zebra's bgpd,handling an incoming update message. . . . 84
6.6 Zebra's bgpd decision process (Simplified). . . . 84
6.7 Small part of the bgpds VTY tree. . . . 85
6.8 Defining a command. . . . 85
6.9 Installing the command in a node. . . . 85
6.10 Inserting the Ghost-Flushing code in the decission process. . . . 86
6.11 This is the VTY node tree with the new commands. . . . 88
6.12 Two LANs, one for the BGP4 traffic y the control Traffic. . . . 90
6.13 Exporting files. . . . 90

2.1 Esquema (Simplificado) de el procesado de un anuncio en la implementación de BGP4 en SSFNet. . . . 104
2.2 Distribución de clases IP 105
2.3 Ejemplo de una configuración de Router Reflectors 106
2.4 Filtros aplicados. . . . 106
2.5 Caótico ejemplo de proveedores de Internet. . . . 107

3.1 Ejemplo de información fantasma creciendo en un escenario de fail-down con un alto E_{down}. 110
3.2 Información fantasma extendiéndose en el escenarion fail-over. . . . 111
3.3 Regla ghost-flushing en su enunciado original en inglés. . . . 112
3.4 Paso a paso la regla ghost-flushing en acción. . . . 113
3.5 Un mal caso para la regla Ghost-Flushing. . . . 114
3.6 Regla Ghost-Buster en su enunciado original en inglés. . . . 115

4.1 Tratamiento de actualizaciones con Ghost-Flushing en el esquema BGP de SSFNet. . . . 118
4.2 Un ruta llega y el protocolo intenta enviarla, aunque el temporizador delta no ha expirado. 120
4.3 Temporizador Delta Timer Expira. . . . 120

5.1 Multi E_{down}. . . . 121
5.2 Mezclando BGP4 tradicional con ghost-flushing. Más es peor. . . . 122

6.1 Estructura modular de GNU Zebra. Todo acaba pasando por el demonio Zebra. . . . 126

A.1 Pseudocode implementation of Ghost-Flushing over BGP. . . . 135

B.1 Example configuration file for a zebra node. Its IP(ID) is 192.168.0.2. It has 5 neighbors. It uses Ghost-Flushing and it shows the debugging messages about it 137

E.1 "29 nodes structure'. Image Generated by RaceWay network visualization tool 145
E.2 "110 nodes structure'. . . . 145
E.3 "208 nodes structure'. . . . 146
E.4 "409 nodes structure'. . . . 146

F.1 Zebra testing Lab, A1203. . . . 147
F.2 Hp procurve switch 2524 used during the tests. . . . 147

Preface

This project/Master's Thesis (Depending on the country) was developed during a Erasmus Exchange Program between the Centro Politécnico Superior of Zaragoza(Spain) and the Luleå Tekniska Universitet (Sweden) during the 2003-2004 course.

I want to thank my thesis director in Sweden, Lenka Carr-Motycková, for her help during this project and for giving me the possibility to work in the interesting world of routing protocols. Also, i would like to thank Jesús Alastruey Benedé, my project representant in Spain, for his advice making possible this project to adjust to my home's university standard.

I must not forget my parents, Gonzalo and Nieves, without whom I wouldn't have the chance to exist, study and go on this exchange program. Thank you.

A special point about the people of the computer support (Specially Johan Hallbäck) , who "supported" and suffered my attempts of killing the sigma sun machines. They were also very helpful with any problem or need I had with my computers.

Finally, I want to greet and thank my friends here in Luleå and back home that helped me to enjoy this great year in a foreign university (Including the guys in the LURC), thanks.

To all them, thank you.

Gonzalo, August 2004

Este proyecto/Master's Thesis (Depende del país) fue llevado a cabo durante un programa de intercambio Erasmus entre el Centro Politécnico Superior de Zaragoza (España) y la Luleå Tekniska Universitet (Suecia) durante el curso 2003-2004.

Quiero agradecer a mi directora de proyecto en Suecia, Lenka Carr-Motycková, por su ayuda durante esta tesis y por darme la oportunidad de trabajar en el interesante mundo de los protocolos de encaminamiento. También, quiero agradecer a mi ponente en España, Jesús Alastruey Benedé,por sus consejos a la hora de hacer que este proyecto se ajuste a los estándares de la Universidad de Zaragoza.

No debo olvidar a mis padres, Gonzalo y Nieves, sin los cuales ni existiría, ni podría haber estudiado o accedido a este programa de intercambio. Muchas Gracias.

Una mención especial para la gente de soporte informático (en especial Johan Hallbäck), que soportaron y sufrieron mis intentos de sobrecarga de los computadores sigma. Me ayudaron además en todo momento con cualquier problema o necesidad alrededor de mis equipos o la red.

Finalmente, quiero saludar y agradecer a mis amigos en Luleå y España, por ayudarme a que disfrutar de este año en una universidad extranjera (Incluida la gente del LURC).

A todos ellos gracias.

Gonzalo, Agosto de 2004.

Part I

Complete Report

Chapter 1
Introduction

In the beginning, networks existed but they were isolated, not connected between them. Then, the need of exchanging information between machines connected to different networks appeared. And so, routers were invented to make this possible. After some time, the interconnected network became bigger and another problem appeared, it was too complicated to manually configure all the routers so they knew how to reach all the networks. As a consequence, some routing protocols were written for the routers to talk between them and exchange reachability information about the networks the knew.

However, the size of the Internet (All these networks connected) didn't stop growing, so, the time came when this routing protocols were not suitable for the Internet anymore. To solve this, a new idea appeared, the Autonomous System. They divided the Internet in Autonomous Systems (AS) and two kinds of new routing protocols were written, external and internal routing protocols. The internal routing protocols were used to exchange routing information inside the ASs and they were like the old ones used before. The external routing protocols had a different mission: to exchange routing information between the routers on the borders of the ASs. It was the 80s and the moment when the Border Gateway Protocol (BGP) was born. BGP is the external routing protocol that became an standard in the Internet.

After some time, new problems came up, so new versions of BGP were developed. In the early 90s, the last version, BGP4, was released and established on the Internet. BGP4 has become the so-called *"glue that holds the Internet together"*, thus, any modification that could improve its behavior will have a positive effect in whole internet. This project runs around improving the behavior and performance of the BGP4 in certain situations.

1.1 Motivation, Purpose and Goal

BGP4 is the external routing protocol (EGP) most used this days. This makes its behavior a crucial issue to the stability of the Internet. Any flaws or problems that could exist in the protocol would have a critical impact in the world-wide communications system [12, 13, 15]. Actually some problems have been found related to the performance of BGP4. The performance and reaction time of the internet are crucial for it to compete with the reliable, fast and low latency phone digital network. Real time services are more and more common these days over the internet and they require a reliable, fast reacting internet.

As some studies have shown, the performance of BGP4 can decrease depending on the topology of the network [12]. In cases when parts of the network were down, it could even take minutes for the rest of the nodes to reconfigure according to the new situation. This is called *convergence problem: the time that it takes until the network stabilizes after an event happened to it.*

Many solutions have been proposed, and between them, we have chosen to study the Ghost-Flushing described in Bremler-Barr et al. [13]. Analyzing its theory base, simulating the BGP4 with this new rule and, depending on the results, implementing it over a routing software are the main goals of this project.

1.2 Structure and method of this work

BGP4 is a critical protocol on the internet, before applying any modifications to the standard, deep work has to be done. A faulty modification can lead to an unaffordable mass failure. This work is the

study/analysis of some proposals to improve this protocol. As a consequence, it was required a proper work structure for the results to be as coherent and correct as possible. Every step is taken after the previous one validates the work done. The steps followed were:

1. Basic Protocol and environment study.
2. Problem study and analysis of the theory behind the solutions.
3. Simulation and comparison of the original and modified versions of the protocol.
4. Implementation of the approved solutions on a real routing software.

1.3 Chapters Overview

This is the organization of this thesis:

Chapter 2 tries to show a picture of how BGP4 works. It summarizes all the aspects about routing and BGP4(Defined in RFC-1771 [28]) that may be needed to understand the work on this thesis.

Chapter 3 contains the definition of the convergence problem studied in this thesis. The rules Ghost-Flushing and Ghost-Buster defined in Bremler-Barr et al. [13] are here shown and analyzed.

Chapter 4 shows how the simulation environment was set. It includes the analysis of the candidate network simulators for this task (Facts, setting up and validation), a small schema of the internal structure of the chosen simulator (SSFNet) and finally, the details about the modification of the BGP4 implementation of SSFNet.

Chapter 5 is a log of the simulations run about the BGP4. Parameters, strategy, cases (Topologies) simulated, the results and their analysis in this section.

Chapter 6 summarizes the implementation procedure: Choosing the routing software to be modified (Zebra), a schema of its architecture and a small analysis of the code and its modifications. Also the testing procedures of the modified software are included.

Chapter 7 contains some reflections about the results of this thesis and their quality.

Chapter 8 shows what further work can be done about this project.

Chapter 2

BGP4 Routing Protocol and its Environment

2.1 Network Routing

2.1.1 Hop-by-hop Routing

Nowadays and in the past, many kind of networks have existed. To make possible the exchange of data between them, the Network Layer of the Network Stack Model was designed. The standard Network level protocol used on the Internet is IP, where each network owns a range of IPs and one IP identifies one host. Then, the role of the routers is to interconnect different networks, if a host wants to connect to another one out of his network, it'll do it through one router that connects the networks. This router will see if it can directly reach the network where the destination is hosted. If not, it'll try to contact another router and then, the process is repeated again until we arrive to the network where the destination host is connected and finally reachable. This is a summary of the routing strategy used on the Internet: ***hop-by-hop routing***.

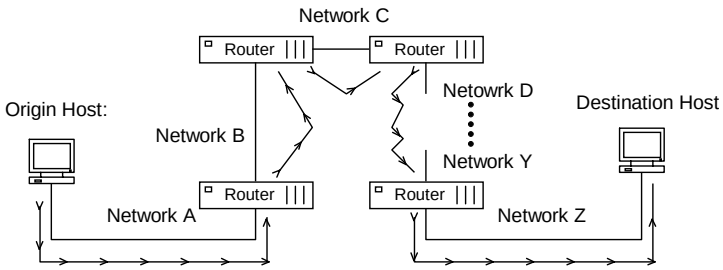

Figure 2.1: Example of hop-by-hop routing between two hosts

By doing this, we simplify the design and configuration of networks. For one network to work and be able to connect to the "rest of the world", two thing are needed: Firstly, a properly configured router connecting the network to other(s) network(s) and secondly, all the hosts having the router as their *gateway*.

2.1.2 Mapping the Internet

The next issue is the configuration of the routers. Let's imagine a network that is connected to three other ones by three different routers. If we want to connect to a host on a remote network, through which one should the connection be established? And even more, each router is connected to our network and three other ones, but, originally, they don't know what are those networks connected to. At the beginning, when the Internet was really small, this was configured statically. System administrators would manually configure each router with the list the networks it could reach by contacting other routers from the networks it had direct connection to. But this model is not scalable. When the Internet began growing manual configuration was ruled out.

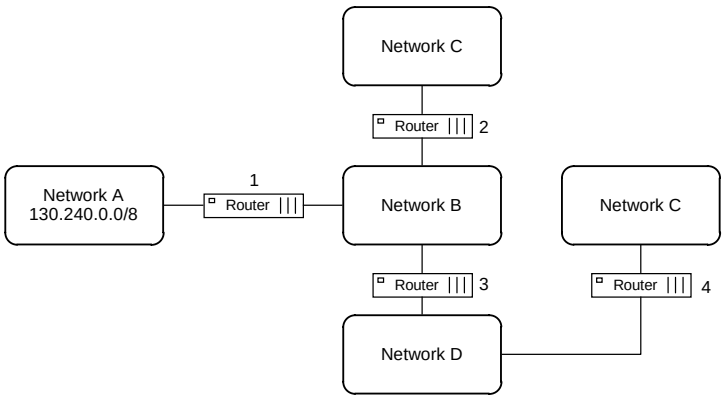

Figure 2.2: Routers communicating reachabillity information.

This was the moment where **routing protocols** were born. The main idea (regardless how is this achieved) is that routers communicate between them, propagating the information about the networks they know how to reach them. Let's picture an small example, a situation like the one on *fig 2.2*. Router 1 is manually configured with the IP range of Network A (130.240.0.0/8). Then, router 1 will contact routers 2 and 3 to tell them that through 1, the IP range 130.240.0.0/8 can be reached. Then, Router 2 and 3 will inform all the routers they know (Except for Router 1) about the new IP range they learnt. For example, router 3 will communicate router 4 that it knows how to reach Network A (Apart from network B). This way the reachability information will be spread automatically.

Due to the different network topologies and situations (Technical and political) on the Internet, routing protocols are not so simple. There are other characteristics that are needed for them to work. Loop detection, fail detection, subnetworks, policies (Filtering), low number of IPs, CIDR support, metrics, links quality, etc. are only examples of things that routing protocols have to support and deal with. Examples of this protocols are RIP, OSPF, IGRP, EGP or BGP.

2.1.3 Exterior/Interior routing

At the beginning of the 80s, all the routers in the Internet shared their reachability information between them. At the same time, the size of the network kept growing unstoppable. This led to the following problems [20]:

- Overhead due to the exchange of routing information between all the routers. Any moment a link would go down that could trigger a ripple of routing messages "flying" all around the network.
- Many routers meant many different softwares, making sometimes fault control impossible.
- Changing versions of the routing softwares was difficult because it would affect the whole network, the model was rigid and inflexible.

So, at this point, it was decided that to split the internet into parts: **Autonomous Systems** (AS) that would comprise a set of networks administrated by the same institution. Also, two kind of relationships between routers was defined depending on their situation: Between routers on the borders of different ASs, *external routing*. Between routers of the same AS, *internal routing*. Routers that were not "border routers" but in different ASs wouldn't exchange information. From this point routing protocols were also divided in **external routing protocols**(EGP), for the border routers, and **internal routing protocols** (IGP), for the routers inside de ASs. The most used external routing protocol has been BGP, and BGP4 is its current standard version. We will proceed to study the basis of this protocol (And thus of external routing protocols) in the next chapter.

We can see now the advantages of the new distribution:

- Failure control and administration became easier.Each AS can deal with its own problems without affecting the rest of the network.

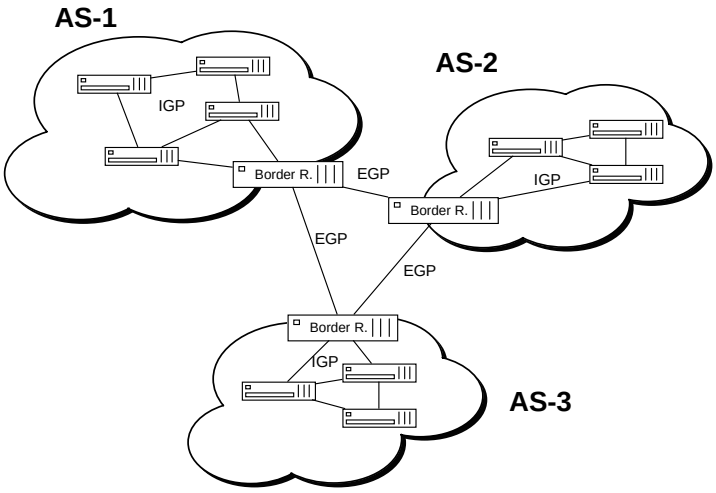

Figure 2.3: New picture of the AS-divided Internet.

- Each network can be owned by different companies so, internally to their AS, they can have their own protocols and policies. The only remaining "standard" is the external routing protocol.

- General software updates of the routers can be done AS by AS, making changes and evolution on the internet easier and thus, cheaper.

2.2 BGP4 Protocol Description

BGP4 is the standard version of the most used external routing protocol used on the internet. The first version was published in June 1989 as RFC-1105. The last version deployed is BGP4, defined in RFC-1771 [28], and is the one studied in the thesis. As shown along this chapter, BGP4 has some features like: loop detection, path vector, CIDR or propagation policies, that allows it to work properly (Or almost) in today's Internet.

This chapter is mostly based on the BGP description on "Routing on the Internet" [20] and the RFC-1771 [28].

2.2.1 Path Vectors and Loop Detection

The approach to the path vector idea in BGP4 is quite radical: Each update message about a route includes a list of all the ASs which that information has traversed through. When this information will be repeated to other routers, the number of the emitting router's AS will be added at the beginning of this AS list. This is the way this path vector is constructed. This list of ASs is known as ***AS Path***.

The path vector has a clear function, **loop detection**. One of the main problem of other routing protocols are the loops in network topology (For example for RIP). However, with the AS paths, a router can check if its AS number is on the list. If it is, it will discard the router, if not, it will accept it, put its AS number and retransmit it.

This technique has also drawbacks. The main problem is that the complete list of crossed ASs about a destination is stored in the memory of all routers and sent on every update message. This means memory cost and message overhead. Some studies [20] state that, with BGP3, 100,000 networks, an average path length of 20 ASs and a total number of 3000 ASs, transmitting a complete routing table would mean 520,000 bytes of bandwidth. Talking about memory occupation for a router, this figures can be exceeded by far. However, this problem would be solved applying CDIR and route aggregation on the BGP4 (This will be analyzed on the following chapters).

2.2.2 Path Attributes

Update messages carry their info as attributes. They can be classified by two criteria, transitive/non-transitive (If it should resent when the route is retransmitted) and well known/optional (Well know means that the attribute must on the update message, and thus understood by all nodes. Optional means the opposite, not compulsory). The main well-known attributes are:

AS Path is the list of transited AS.

Origin if this route is internal or learned from out of the AS.

Unreachable mark , if active, it means that the destination is not reachable.

Inter-AS metric .

Next-hop the node where the path to the destination begins.

2.2.3 The Protocol

To send the protocol messages, a TCP connections between the BGP4 nodes is used(Being the default port used the 169). It relies on this protocol for the error control, making everything much simpler. This decision was criticized due to the original versions of TCP sensibility to congestion, however, modern implementations of TCP have features correcting this problem (slow start, congestion avoidance).

The messages sent by this protocol are of four types: **open, update, notification and keepalive**. This are their function:

Open messages are used at the beginning and establishment of the session between two speakers. It's the first message in the session and it contains information like:

- The AS number of the sending router.
- Hold time, used by the "keep alive" procedure.
- An identifier, the IP of one of the interfaces of the sending router.
- Authentication information and code, specificating the authentication method and keying information.
- The version of the BGP.

An acceptance of this message is done by sending a keep-alive message back.

Updates , once the session is established, the routing information is exchanged using update messages. One update message only contains information about one path. That information is:

- Networks reachable with this path, also known as NLRI (Network Layer Recheability Information)
- AS Path, the ASs traversed by this path.
- Origin of the information, internal to this AS or an external AS.
- Neighbor, who is sending this message.
- Inter AS-metrics, information used to the route choice.
- Unreachable, indication if the pointed networks are not reachable anymore. procedure.

When an update message is received, the information is processed by the protocol. First, the AS-path is checked, if the our AS number is in the path, this update is discarded because it contains a loop. If everything is ok about the update and it announces a new route, the information is inserted in a table called RIB-in. There is one RIB-in per "neighbor", the local node has established a BGP4 session with. If the update is a "withdrawal", the info about that destination is removed from the RIB-in corresponding to the neighbor which sent the update message. After any change in this tables, the "decision process" is run, **evaluating the routes** in all the RIB-INs and **choosing the best ones** (If any) for each destination known(For every destination known) to **put them**

(Or remove) in another table called LOC-RIB (There is only one). This structure contains the routes currently chosen (And used) for every known destination.

After the LOC-RIB has been modified, maybe it's needed to send new information to the neighbors of this node. Another decision process is run, and this time, it chooses what routes should be advertised or withdrawn to our neighbors. This new advertisements or withdrawals ar put in another table: RIB-out. Later, the info in this table is processed and the corresponding updates will be send.

Keep-alive are used by BGP nodes to inform their neighbors that the node is running and there is no problem. If one node doesn't receive a keep-alive message in the hold-time negotiated in the open message, it will try to know if the remote node works by notify messages.

Notify are used to notify errors surrounding the BGP protocol.

There are other characteristics surrounding the messages. The most important are:

MRAI timer or Minimum Router Advertisement Interval. This is a limitation to control the overhead and behavior of the BGP4. In the standard configuration, *per peer*, one node cannot send more than one update message with a new route (advertisement) per neighbor, per MRAI. It can be also configured *per destination*. Then, the limitation is not to send more that one advertisement per NLRI to a neighbor per MRAI. The general recommendation specified by the RFC says that this timer should be initialized to 30 seconds.

Jitter Al timers, keep alive, MRAI, start-up, should apply a random jitter to avoid problems due to synchronization, i.e. open messages collision or other situations due to the network disposition.

Decision Processes In the RFC-1771 [28], three different decision processes are defined. The first one is the one which evaluates the update message just received and will modify the corresponding RIB-in if needed. The second decision process is in charge of reviewing the content of the RIB-INs and modify the LOC-RIB if needed. There can be three reasons why this can happen: Because all the references to a certain destination have been removed from the RIB-INs so the route on the LOC-RIB must be removed. Other cause is that there is a better route in the RIB-IN that the one in the LOC-RIB about a certain destination. Finally, if a neighbor which announced a route that was put in the LOC-RIB makes another announce about the same destination (Implicit withdrawal) then, the new route must be put on the LOC-RIB. All this processes have to execute in mutual exclusion.

Implicit withdrawal , if one node learns a new path about a destination that substitutes the last one it announced and the MRAI timer is not expired, it won't send a withdrawal. Instead, it will wait until it can send the new path. This is called implicit withdrawal.

Choosing the best route for a destination is done in the second decision process. It depends on the implementation and the policy of the institution administrating the AS, but an example of the choosing criteria can be (By importance order):

- AS Path Length.
- Metrics of the different Paths.
- The Local preference. If there are many ways to reach a destination, the destination can prefer to be accessed by one path instead of other.
- Other ones the administrators would want to implement.

Policies are also something to be considered. Even if different ASs are physically connected, and there are sessions between their BGP routers that doesn't mean they will exchange the routing information about all known destinations. This is what is called the AS relationships [18]. The Internet is a network owned by companies with economical interests. Companies charge for carrying traffic or provide connection to other networks. For example, One AS would like to allow traffic from a neighbor to a certain destination but ban it to other (Maybe because this neighbor has hired the company owning this AS to provide connection only to a set of destinations).

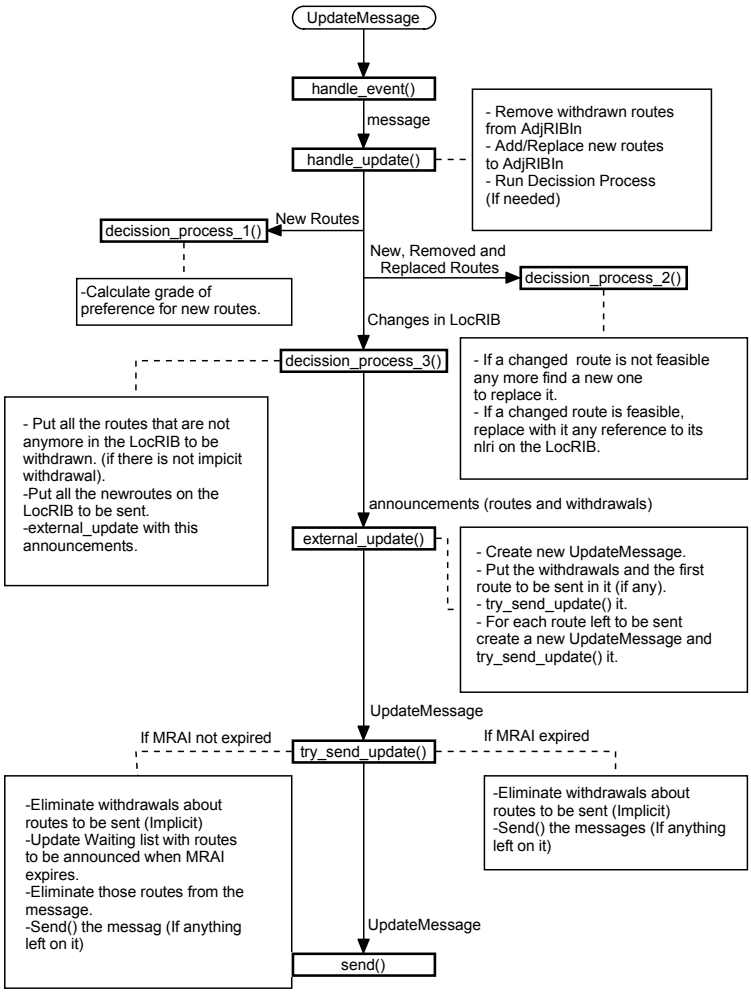
UpdateMessage
handle_event()
message
handle_update()
- Remove withdrawn routes from AdjRIBIn
- Add/Replace new routes to AdjRIBIn
- Run Decission Process (If needed)
New Routes
decission_process_1()
-Calculate grade of preference for new routes.
New, Removed and Replaced Routes
decission_process_2()
Changes in LocRIB
decission_process_3()
- If a changed route is not feasible any more find a new one to replace it.
- If a changed route is feasible, replace with it any reference to its nlri on the LocRIB.
- Put all the routes that are not anymore in the LocRIB to be withdrawn. (if there is not impicit withdrawal).
-Put all the newroutes on the LocRIB to be sent.
-external_update with this announcements.
announcements (routes and withdrawals)
external_update()
- Create new UpdateMessage.
- Put the withdrawals and the first route to be sent in it (if any).
- try_send_update() it.
- For each route left to be sent create a new UpdateMessage and try_send_update() it.
UpdateMessage
If MRAI not expired
try_send_update()
If MRAI expired
-Eliminate withdrawals about routes to be sent (Implicit)
-Update Waiting list with routes to be announced when MRAI expires.
-Eliminate those routes from the message.
-Send() the messag (If anything left on it)
-Eliminate withdrawals about routes to be sent (Implicit)
-Send() the messages (If anything left on it)
UpdateMessage
send()

Figure 2.4: Schema (Simplified) of the Update handling by SSFNet BGP4 Implementation.

2.2.4 CIDR support and BGP4

The Internet behaves sometimes like a living being, it tries to grow as much as possible and as fast as possible. As a consequence, scalability problems show themselves sooner than expected, threatening to lead the whole network into a chaos. This was the case of the IP system in the early 90s. When IP was designed in the 80s, 32 bits looked enough for all the hosts on the Internet, being the first 8 bits the identifier of the network and the las 24 bits the address for the host. However, after some time the three classes for the IP addresses were added (A,B,C). IP range was split between those classes and the number of bit belonging to the network and the host depended on the class. But in the early 90s, this design started to show problems. In the first place, the B addresses were about to exhaust and also, the routing tables where growing too much. A "patch" to the IP design was developed in that moment (Giving time until the next version of IP,IPv6), the "Classless Inter-Domain Routing" (CIDR).

Class B Exhaustion and Routing Table Explosion

As we can see in the figure, A class networks were too big, and C classes too small, so B class networks became quite popular, too popular in fact. The moment came, when almost all the B class networks were assigned, still, more were demanded. C classes were too small for most companies and there were only 256 A class networks, so a real danger of address exhaustion appeared.

Also, the number of networks connected increased constantly making the routing tables bigger inside de routers. This had a negative impact on memory usage and processing time (Storing and searching in

Class	Network Bits	Host Bits	Hosts per network
A	8	24	16,777,214
B	16	16	65,534
C	24	8	254

Figure 2.5: IP classes distribution

bigger tables). This could affect performance of routers and response time to failure. This appeared as another risk to the integrity of the internet.

Classless Addressing, Route Aggregation and BGP4 as a Interdomain-Routing protocol

Classless addresses have a characteristic, they are composed by the whole IP number and a network mask. The network mask indicates which bits (from the IP address) belong to the network address and which ones belong to the host address. Using this we can divide A networks in smaller ones and group contiguous C networks to make bigger ones. Also you can define networks adjusting the size to the demands of the customers who contract the IP ranges. This way, the IPv4 address problem was patched until the coming of the long delayed IPv6. Exchanging routing information between different domains that are not class abiding is the so called CIDR.

Until 1992 there was no relationship between the IP numbering and the actual structure of the internet. But then, a new strategy of "provider addressing" was imposed. From the top levels, the network providers owned a range of IPs, and their customers would have to be in the IP range of the providers. This means a great advantage to routing. Let's compare the old scenario with the new one. In the old system, we have two destinations, with IP range IP1 and IP2. They both connect to the internet through the same provider but their IP ranges didn't have to have anything in common. Than meant two entries in a routing table to locate them. But in the new system, if both are under the IP range of the provider, a router only has to learn how to reach the provider. Then, the provider's routers will redirect the connections to the proper network. Considering that a provider has many more than two customers, we can see the positive impact in the routing tables all over the Internet.

What has this to do with BGP4? BGP3, the previous version of BGP4, didn't support the CIDR or the "supernetting", this is probably the most noticeable difference between both versions. BGP4 has the capacity of aggregating routes that share parts of their addresses. Since the deployment of BGP4 on the Internet, both problems were greatly relieved. Also, since BGP4 knows which ASs the paths go by, it can do special calculations to consider if it's worth to aggregate destinations under a bigger network. Actually, there is an attribute defined on the RFC-1771 to specify if a route can be aggregated or not.

2.2.5 Internal BGP (IBGP)

Although this work has to do with issues related to the external BGP4, we cannot keep going without mentioning the internal part of the protocol. BGP4 can also be used between routers inside the AS instead of using other possibilities like OSPF or RIP (Even if BGP4 and this protocols can co-exist). The initial idea was to establish a full meshed BGP4 network between the routers inside the AS and the border routers. This way, the routes are disseminated all over the AS, but in a quite inefficient way. So other options were proposed, some examples are the Router Reflector [30] or the Confederations (Also applied to external BGP) [25] approaches. They try to reduce the number of BGP sessions established between routers.

The **Route Reflector**[30] approach is based on the idea that we can divide an AS in clusters (internally the clusters are full meshed). The routes are provided by a *route reflector* to the whole cluster. This route reflector is just another BGP4 router that is connected to the border routers (Can be a border router itself) and other route reflectors.

Other possibility are the **Confederations**[25]. The idea is to group various ASs in a virtual bigger one, or to divide a big AS in smaller virtual ones (The external or the Internal BGP approach). This way we avoid the connections derived from a full meshed model.

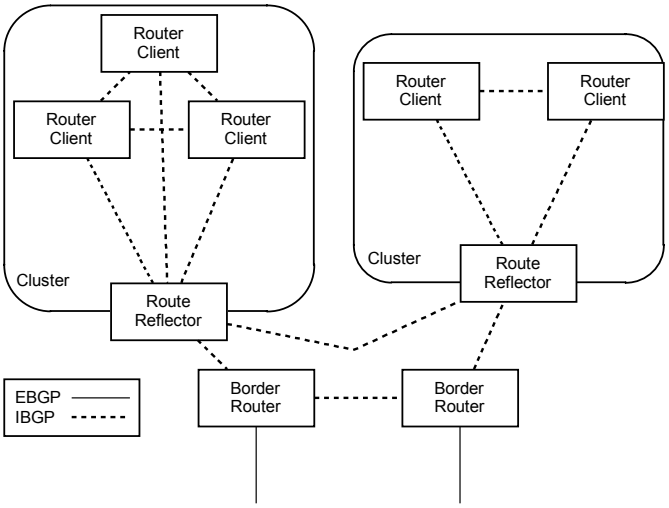

Figure 2.6: Example of a Route Reflector Configuration inside the same AS.

2.3 Autonomous System Relationships

This work is about the problems of the BGP4 in some topological situations, so, in this chapter, we will try to show some examples of the most typical BGP4 relationships that found on the internet (As showed on the article "Inferring Autonomous System Relationships"[18]). This chapter highlights parts form that article relevant for this work. In the next chapters we will relate this examples with the problematic situation.

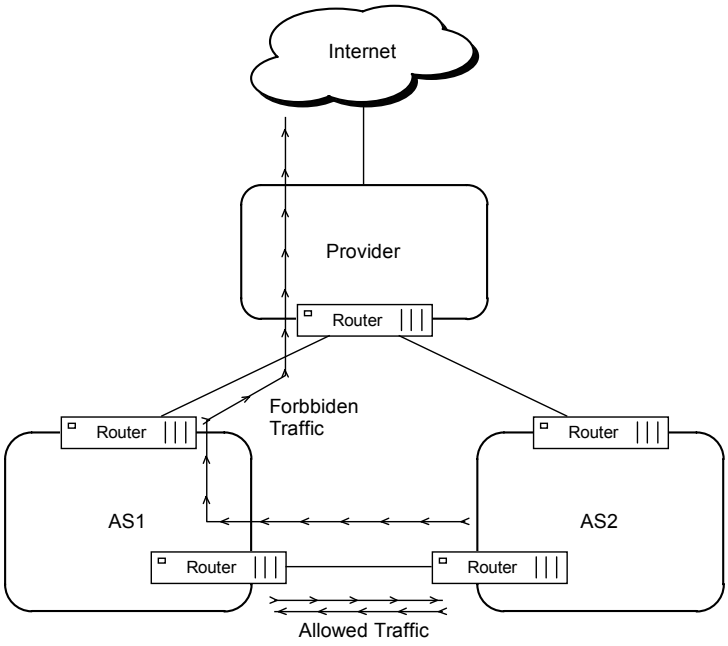

Figure 2.7: Filter Policies.

This relationships exist due to various reasons, first the **topology of the network**, how the different ASs and networks are physically connected. The other factor are the **peering policies** established between the ASs due to economic and commercial relationships between the institutions owning the ASs. The peering policies take effect in the filters set over a BGP session. I.e. two ASs have hired

access to the Internet through the same provider but they are also connected directly (*fig. 2.7*). One of them would like to access the other but not to allow the access the internet throw itself. These are the commercial reasons we are talking about.

2.3.1 Relationship Classification and Internet Disposition

Exporting to a Provider : in exchanging routing information with a provider, an AS can export its routes and its customers routes but usually does not export its provider or peer routes.

Exporting to a Client : In exchanging routing information with a customer, an AS can export its routes and its customer routes, and as well as its provider or peer routes.

Exporting to a Peer : In exchanging routing information with a peer, an AS can export its routes and its customer routes, but usually does not export its providers or peer routes.

Exporting to a Sibling : In exchanging routing information with a sibling, an AS can export its routes and routes of its customers, and as well as its providers os peer routers.

Basically, the internet is a collection of nodes structured in pyramidal levels but with a collection of links between nodes cross-jumping levels. Some ASs are part of the backbone structure, this ones provide other ASs with interconnection. This other ones are also providers to smaller networks, and then it repeats itself until reaching the access-host in the edge of the network. BGP4 is the exterior routing protocol that exchanges routes between this levels. Actually, all the ASs can be highly interconnected, and this relationships are used to control the traffic between them. This high interconnection has a high impact on the problem studied on this work.

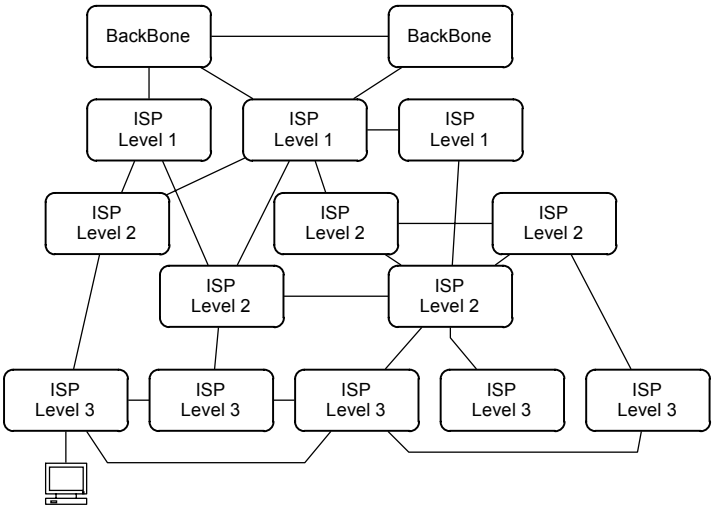

Figure 2.8: Chaotic example of Internet providers interconnection.

2.4 History

In this chapter we will try to make a small summary of the Internet time-line highlighting those events related with the evolution of routing protocols(Based on *Hobbes' Internet Timeline* [37], *Connected: an Internet Encyclopedia* [2], *The Wikipedia*[11] and various RFCs[28, 29, 19, 24, 21, 22, 32, 35, 27, 21]).

1969 ARPANET is born. Very few nodes. First Dynamic routing protocols used in flow distribution.

70s ARPANET keeps growing, IP is not invented yet. So routing protocols are not standard, access level class or static-manual definitions.

1982 TCP/IP is defined and applied to Internet. Internet as an interconnection between networks is born. EGP (RFC-827[29]) specified and used for gateways between networks.

1983 DNS is born and started up.

1986 NSFNET, the first back bone (56 kbits) is born.

1988 First specification of RIP established (RFC-1058[19])

1989 100,000 hosts barrier broken. OSPF Defined (RFC-1131[24]). BGP defined (RFC 1105[21]).

1991 BGP-3 Definition (RFC-1267[22])

1992 Topology related IP addressing. Supernetting proposed (RFC-1338[32])

1994 CIDR begins to be deployed. BGP-4 Definition (RFC-1654[35])

1995 NSFNET reverts back to a research network. Commercial interconnected networks become the backbone of the Internet. Proprietary protocols are developed by companies like Cisco or Juniper networks. Last BGP-4 standard released (RFC-1771[28]). Also first standard for IPv6 released (RFC-1884[27]). (RFC-1884).

End of the 90s to now The breathing space given by CIDR, supernetting and BGP4 is running out, IPv6 is started to be deployed but it doesn't reach all the internet.

2006 Supposed date for IPv6 final stages of deployment in some asian countries (Korea, Japan...) [11].

In summary, everything began with isolated networks that didn't exchange information. Then, in end of the 70s, exchange began, being the establishment of TCP/IP in 1982 the big first step to standardization. From that point, the routing protocols as we know them were born. Many have appeared but all of them were scientific initiatives or industry products that imposed themselves on the network. The 90s are marked by the urgent need of solving the problems derived of the IPv4 structure, CIDR, Supernetting and BGP-4 were born then. When IPv6 is finally deployed new changes in the Internet routing schema are expected.

Chapter 3

Ghost Flushing and other Proposals

As we have stated before, BGP4 is a routing protocol that has solved many problems over the Internet. But it suffers from various problems derived of the many different environments (Topological and political) in which it runs: A too open standard that allows too different implementations[31], route instability[16, 33], strange behaviors due to misconfigurations[12], bad interactions between internal and external BGP[33] or too high convergence time[12, 13, 15] in certain situations. In this work we are analyzing a small set of the problems related with the high convergence time. This chapter will define precisely the scope of this work and present the solutions proposed to solve this set of problems. Besides, an analysis of the theory behind them will be done.

3.1 The Convergence Problem

3.1.1 Some initial concepts

Before getting inside the problem, a set of common concepts related to the problem have to be fixed.

Networks will be seen as graphs formed by nodes (BGP4 speakers) and arcs linking them (BGP4 session between two speakers).

Event Whatever can happen in the network (i.e. a link down/up, a node down/up) that would alter the network. I.e. Fail-down, fail-over, system-up, shorter-path.

Fail-down A destination is not reachable anymore.

Fail-over A destination is now reachable through a longer path than before.

System-up A destination that was unavailable is now reachable.

Shorter-Path A destination now is reachable through a shorter path than before.

Convergence Time Time since an event happens on the network (Fail-down, fail-over, system-up, shorter-path) until the configuration of the network stabilizes and the routing tables of all the nodes reach a final state (It won't change until another event happens.

E_{up} Convergence time if system-up event happened.

E_{down} Convergence time if fail-down event happened.

E_{longer} Convergence time if fail-over event happened.

$E_{shorter}$ Convergence time if shorter-path event happened.

3.1.2 High Convergence Time and The Ghost Information

As shown in some articles [12, 13, 15] there is a strong relationship between the network topology and the convergence time. Also, **one factor that is critical in the convergence time is the MRAI timer**. After any change in the network, there is always the limitation for MRAI seconds between two

updates sent from one BGP speaker to same one. If many updates are needed, this time accumulates, being the convergence time the result of this "waits" for the MRAI timer.

The proposals studied in this work and extracted from *Bremler-Barr et all*[13] are based also in another idea: *One lie makes many and in networking it continues recursively.* This means that if a node makes a decision based on information that is no longer valid, the updates sent by this node can contain information won't valid either. Afterwards, other nodes will make decisions based on false information, and then on and on. Some time after, this false information will be flushed from the network, but it will take more messages, more updates and more time. If we add the delay between updates due to the MRAI limitation, it's easy to see that this can be the origin of a high convergence time situation.

After this example we can then define the idea of **Ghost Information**[13], route-table contents, assumptions or messages generated based on not-valid-anymore information. According to the classical model of BGP4 this ghost information would generate more ghost information and then on and on.

Let's picture an example to show the impact of the **ghost information in a fail down situation**[13]:

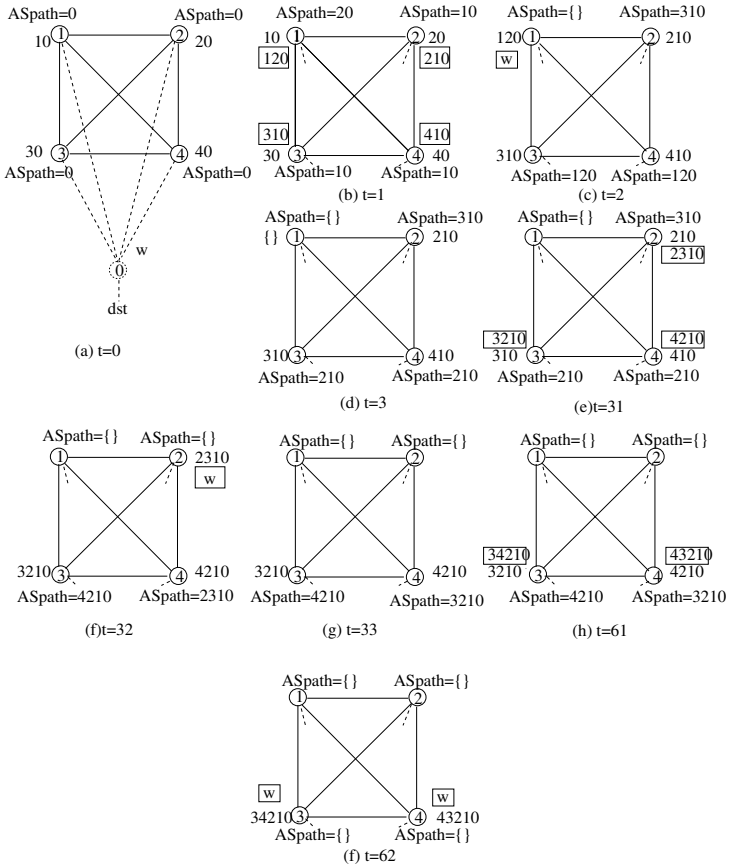

Figure 3.1: Example of Ghost Information spreading in a clique topology and high E_{down}.

In *fig. 3.1* we can see a clique topology network, 5 nodes full-connected. To understand the notation, the numbers inside the nodes are the AS numbers they belongs to. The numbers on the sides are the last AS path announced by each node about the destination to node 0. The numbers inside the squares are the AS paths that will be announced in next round, a "w" means a withdrawal. The AS path that is using in that moment by each node is the "ASpath=" indicator. We presume that the network is synchronous, where events happen in 1 second rounds. If something is sent from one node to the other, it will arrive in the next round, 1 second later.

Now, analyzing what happens in this example. In t=0, all nodes can reach node 0 directly, but then this nodes goes down. In the next round all nodes choose a differen path and want to announce it. Let's look to a single node as an example (Node 1). In t=1,Node 1 knows that the other nodes can access 0, so it chooses de best one. Since all paths are the same (Quality) it chooses the one with the lowest id (2). Then, node 1 will announce the new path it is using 1-2-0. This is the first example of ghost

information. Node 1 doesn't know that node 2 cannot access 0 either, so it's making assumptions over information no longer valid. Now let's keep looking at what happens later. Meanwhile, the rest of the nodes have done the same but choosing 1 as the next hop to reach 0. Node 1 will learn about this in t=2, and send an immediate withdrawal to everyone. However, it is too late, in t=3, all nodes realize that 1 cannot reach 0 and choose another AS path. But all they sent an update a second ago, so they have to wait MRAI (30 seconds) until they can send a new update. After 30 seconds the process repeats itself, but this time node 2 will be chose as the "jump point". Another 30 seconds.

In this example we can clearly show how the combination of a **highly interconnected topology** (clique), **MRAI limitation** and **ghost information** have as a consequence a extremely high convergence time, 60 seconds. A time far too high for some real-time services that run over the internet. So this is the problem this work studies, from all the high convergence situations that BGP4 suffers, this is the one we will analyze and try to correct following the Ghost Flushing article [13].

Another case worth examining is **the ghost information and the fail-over situation**. Let's show an example. As we can see in *fig. 3.2*, the ghost information is not flushed until the new path traverses all the nodes of the alternate path.

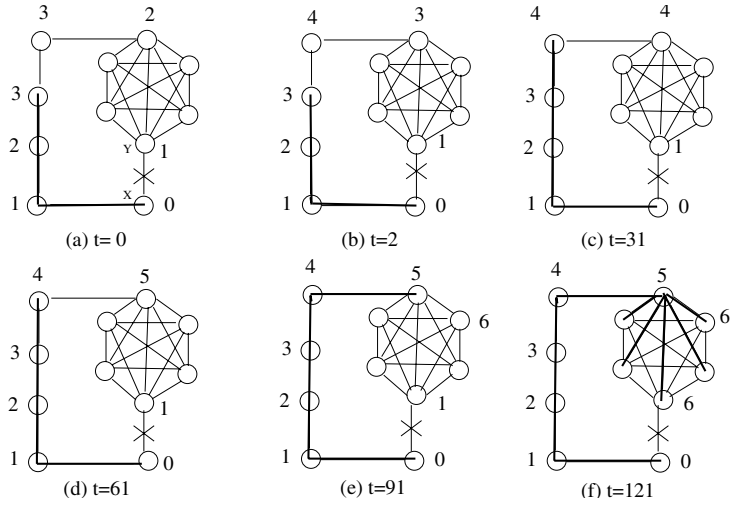

Figure 3.2: Example of Ghost Information spreading in a fail-over scenario.

3.2 Ghost Flushing Rule

3.2.1 How it works

The idea behind the ghost-flushing rule is to try to eliminate all the ghost information as soon as possible. This modification of the BGP4 protocol wants to avoid any changes to the messages used by the protocol (May cause incompatibilities) or any high cpu/memory operations to be done. So it relies on simple rules to work. In *fig. 3.3* the enunciate of the rule can be found. In the functioning of this proposal two clear tasks can be found:

Identifying Ghost-information If the path to a NLRI is updated to a worse destination, it means that the old path is no longer valid, it is ghost information. We may have announced this path before, so ghost information exists.

Action to be taken The first option is to send immediately a new update containing the new path (Implicit withdrawal). But this may be not possible because of the MRAI limitation, however, a withdrawal still can be sent, *flushing* the ghost information generated by us that is not longer valid.

The withdrawal messages have a different meaning from the one that classic BGP4 gives. Before, a withdrawal message meant that a node doesn't have any path the withdrawn destination. Now,

under the ghost flushing rule, a withdrawal message means that the previous AS path advertised about a certain destination is no longer valid. A router may have a route to a destination and still send a withdrawal about it.

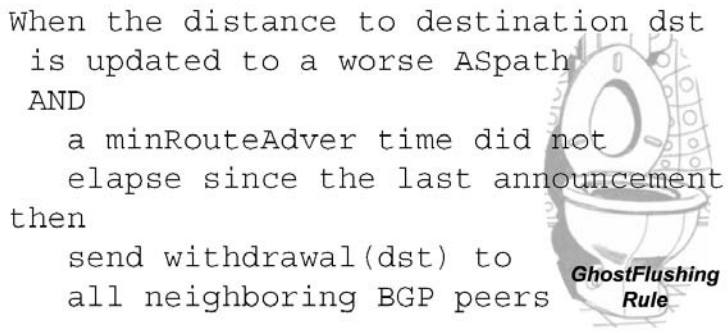

Figure 3.3: Ghost Flushing Rule.

Now let's show in an graphical example how it works. In *fig 3.4* the same situation as in *fig 3.1* is shown, the five nodes clique. Until t=2, every nodes does the same as with the traditional BGP. But then, all the nodes (But node 1) updates their path to go to 0 for a longer one, but they cannot send a new update because in t=1 they already sent one and MRAI is not expired yet. So the ghost-flushing rule comes into action and a general withdrawal is sent because ghost information has been detected. As a consequence, in t=3 all the nodes empty their routing tables and they stop taking any action. Just in 3 seconds.

Looking to this example we identify two phases:

Ghost-Information spread is born because of the first implicit withdrawal. But this longer routers trigger the ghost- flushing rule in all the nodes, in the example it would happen between t=1 and t=2.

General Flushing because of the ghost information detected. All nodes clean the ghost information they hold in their tables. Happens in t=3.

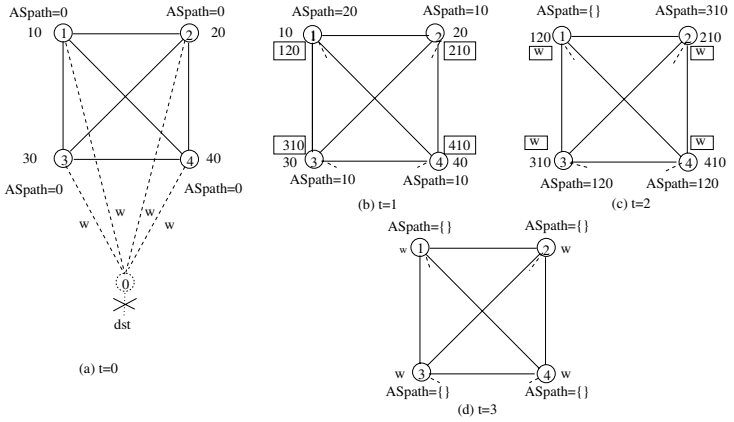

Figure 3.4: Step by step, Ghost Flushing Rule in action.

3.2.2 Convergence time

In [13] there is a deep mathematical analysis and complete demonstrations of the results that are going to be shown in this chapter. Here we will show the results of this analysis and summarize the idea behind them. The notation used must be specified:

h link latency.

n number of nodes.

E number of links.

minRouteAdver minimum route advertisement interval(MRAI).

Now we will show all the convergence times relevant to this work.

Fail Down situation

$$E_{down} = n * h \tag{3.1}$$

$$messages = \Theta(\frac{2hnE}{minRouteAdver}) \tag{3.2}$$

For the E_{down} the idea is simple. After k rounds all AS paths shorter or k long have been flushed. At the beginning, the closer nodes to the failure will flush their routes and this flush will propagate, eliminating longer paths that farder nodes hold. In the worst case, the fardest node is n hops away(h seconds every hop), so it finishes being $n * h$.

We can see that the complexity is much better than the one on the traditional BGP with $minRouterAdver*$ n.

Fail Over situation

Analytically is not possible to prove that the ghost flushing is always better that the traditional BGP in the Fail Over case. As pointed in [15], the convergence time will depend on how long it takes until the ghost information vanishes and how long the new path propagates. We will see in further chapters that the case showed in 3.2, improves using the ghost flushing rule. However, there are extreme cases where the ghost-flushing BGP behaves worse than the traditional BGP . An example can be seen in *fig. 3.5.* The chain of events in the figure looks like this:

- Node X updates from the path through K to the path through M. And it is announced to S.
- Then, X realizes that path is not good either. So now it chooses the one through Y that is longer.
- The last event triggers the Ghost-Flushing rule, sending a withdrawal to S.
- X sends the new good route to S after MRAI time.

With the ghost-flushing rule, this case turns out to behave worse because S should always route through X to reach Dst. But during MRAI time it won't have that route because of the withdrawal generated by the ghost-flushing rule. We can state that *this will happen always that more than two alternate paths spawn from the same node and at least two fail at the same time, being the third one longer.*

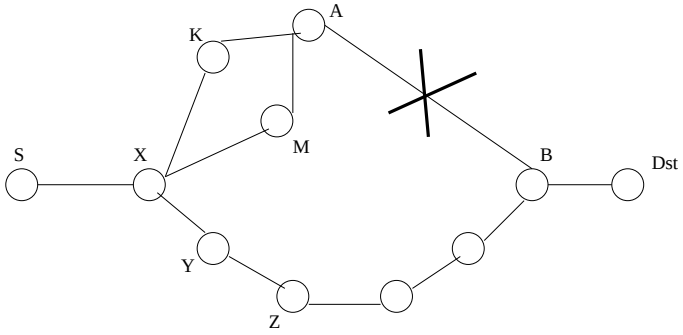

Figure 3.5: Bad situation for ghost-flushing.

Start-up and Shorter-Path

For this cases, no update to a longer path will happen on the routing tables of the routers. This means that the ghost-flushing rule won't be triggered, thus no difference with the traditional BGP will happen.

3.3 Ghost Buster Rule

3.3.1 How it works

In the article [15] another rule was defined to improve the behavior of BGP on the studied case. It is the **Ghost-Buster rule**. This rule is supposed to run over the ghost flushing rule, both working at the same time. The idea that originates it is that, while withdrawals work as a cleaning, the updates could be delayed even more so the ghost information is blocked completely. This more aggressive approach, stopping the ghost information, stems from two reasons. In the first place, some implementations don't include the MRAI limitation, so this way, we make sure the timer is used. The second reason is to try to model the interaction between the ghost-flushing and some route oscillation strategies like routing damping mechanism[16], that can mean bigger delays to the spread of update messages than the original BGP.

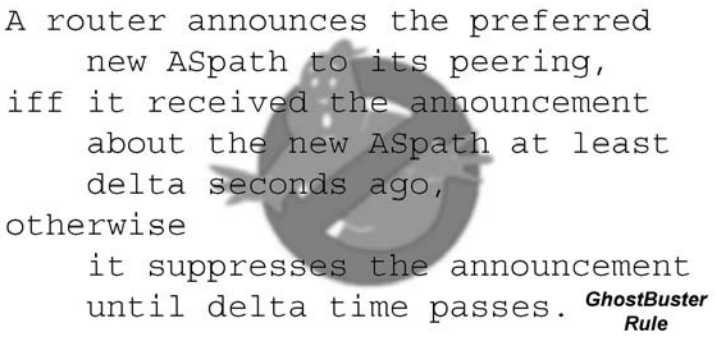

Figure 3.6: Ghost Buster rule.

The delta value can be set with any value, however, as the simulations will show, if it's less than the MRAI value, no effect of this rule is noticed (Most cases). Again a more complex definition and mathematical descriptions can be found in [15].

3.3.2 Convergence time

Fail Down situation

Being $K = \frac{delta+h}{h}$

$$E_{down} = hd\frac{K}{K-1} \tag{3.3}$$

The idea behind this formula is that according to the ghost-buster rule, the ghost AS path in the network can increase only once every $delta + h$ time. The maximum length of an AS Path is d, then, the equation for t (The convergence time) is:

$$d + \frac{t}{delta + h} = \frac{t}{h} \tag{3.4}$$

By definition of K we can replace $delta + h = Kh$ and get:

$$d + \frac{t}{Kh} = \frac{t}{h} \tag{3.5}$$

and then:

$$t = E_{down} = hd\frac{K}{K-1} \tag{3.6}$$

Other situations

This rule requires *all advertisements to be delayed*, this means a negative impact on other situations like start-up, shorter path or fail-over. The interest of this rule is double, to clearly understand the convergence properties of the BGP and to see the possible interaction of Ghost Flushing and some path oscillation control techniques.

3.4 Other Solutions

3.4.1 Reset Rules

The reset rules [15] extend the ghost buster rule by applying the delta timer from any kind of update, advertisement or withdrawal. It doesn't have any applied interest in the real world so it wasn't further analyzed.

3.4.2 Other Approaches to the same problem

Reverse Poisoning (IGRP) It's a non scalable situation, it first cleans al the routes and then it begins again. It is suitable for internal routing, but definitely not for external.

Route Consistency Assertions to truly identify the ghost information requires extra computational time that would harm the general performance of the router. Also show heavy problems over AS partitioning [13].

Active loop detection on sender [15] Reduces the convergence time, but as simulations will show, ghost-flushing behaves better.

Proposal on [12] means adding extra information to the withdrawal messages. And the message system was something we didn't want to modify.

Chapter 4
BGP4 Simulation Environment

The simulation part of this thesis meant much more work than just running a set of scenarios. In the first place, a simulator had to be chosen, that included: looking for a list of available simulators with BGP4 functionalities, analyzing them, learning how to use them, creating some examples and testing the results. After that, a proper simulation environment was needed, programs to automatically generate the configuration files, run the simulations and analyze and store properly the results. In this chapter we will try to show how all this was done.

4.1 Choosing a Simulator

Of all the network simulating suites available we chose to test two: NS-2[7] and SSFNet[9]. There were other options like J-Sim[5] (Formerly know as JavaSim) but due to time limitations we had to stick to testing the two more well known simulators (With the best reputation).

4.1.1 NS-2 and BGP++

The first option was NS-2. NS-2 by itself doesn't include BGP4 capabilities. But there is an extra module called BGP++[1] that is an adaptation of the C code from Zebra[4] routing software. Its Zebra relationship was the strongest point for this option, because Once the implementation of BGP++ was known, modifying Zebra would be much easier.

History and characteristics

Ns began as a variant of the REAL network simulator in 1989 and has evolved substantially over the past few years. In 1995 ns development was supported by DARPA through the VINT project at LBL, Xerox PARC, UCB, and USC/ISI. Currently ns development is support through DARPA with SAMAN and through NSF with CONSER, both in collaboration with other researchers including ACIRI. Ns has always included substantal contributions from other researchers, including wireless code from the UCB Daedelus and CMU Monarch projects and Sun Microsystems.

NS-2 is a event driven object oriented simulator. It has two differentiated parts, the simulator engine and the model definition interface. The simulator engine is developed in C++ trying to achieve a high performance but, at the same time, keep an easy to understand and modify structure. Meanwhile, the simulation interface is implemented in OTcl (Object Tcl) therefore the simulations are configured in OTCL. This is also a strong point favoring this simulator. Tcl is a good scripting language to develop programs to run the simulations and configure them. Since the configuration interface is a suite of tcl commands that manipulate the internal simulating classes all the simulating process can be written and developed fully in TCL.

BGP++ is quite similar to Zebra. Configuring the BGP nodes is done like in real Zebra nodes. You can write configuration files that would run on the Zebra software (It actually uses a notation cloned from Cisco)[1] and feed them to the simulator. After, it generates a log file per each BGP node which content is fully configurable: from state machine logging to table dumps after every update.

[1] An example of the Zebra/BGP++ configuration files can be found in the Appendix

Setting NS-2 and BGP++

The TCL-C++ combo is also one of its weaknesses. Since this simulator runs over C++, it has to be compiled in the machine it is going to run on. Also it only works properly with a set of versions of the applications it needs (TCL between them). We tried to make this program run on a department time shared Sun machines, but the version of Tcl that came with Solaris 9 was not compatible. Changing the TCL version wasn't an option (It is a department administrated machine). So we had to set up a stand alone machine just for ns-2 (A much slower one, and with much less memory). After successfully compiling the ns-2, it was moment to add the BGP++ module. The installation included to apply a patch to the NS-2 simulator and then, the first problems appeared. It wasn't compatible with the latest version of NS-2, so we had to jump back to an older version of ns-2 and start again. After applying the path Ns-2 had to be recompiled, but it took some extra time due to some problems between the patch and the simulator. After many unsuccessful compilations we made them work together.

Then it was the moment to learn how to use the simulator. We found a good tutorial[6] and the first thing noticed was that there were some bugs in a ftp transfer simulation (It was part of the tutorial). We reinstalled the software and tested the example without the BGP++ patch and the problem disappeared. Since everything else worked finely (Seeemed so) and for the BGP simulations we didn't need the elements, we applied again the patch (Painfully again) and started the testing part.

To create the simulation scenarios (And trying to look into the future) some small programs were developed in tcl to create clique scenarios of different sizes. They intended to reduce the time of simulations set up. Even if this simulator wasn't used in the end, this software was revamped to create the configuration files for the GNU Zebra bgpd, so it will be described in further chapters.

Testing examples

The first test that we decided to do was to simulate the five node clique situation on *fig 3.1* and evaluate how close to the reality were the results. After setting up the network scenario, starting the simulation and getting the log files of each node, we first noticed that, against what [13] says, the situation converged in 1 or 2 seconds. Looking deeper into the code a series of strange behavior for the BGP4 were found. The most disturbing one was that it didn't respect the MRAI limitation imposed by the BGPs RFC[28]. We got in contact with the developers of BGP++ and they sent us a patch for the module. After recompiling all the software together, the simulations started again. [2]

Still, the simulations showed that the simulated case(*fig. 3.1*) converged in a much shorter time that it was supposed to. Again, a deeper analysis of the logs was done. And another strange behavior came to our attention, it looked like after a node would receive a withdrawal about the information it used to make its last updates, it would send also a withdrawal. This behavior was far too different from the one specified on the RFC[28].

After all this problems, this network simulator was discarded. A difficult set up, the lack of possibility to run the same compilation in different machines at the same time (C++ based) and this final problems of not standard behavior in the BGP model, pushed this simulation suite out of the project.

4.1.2 SSFNet 2.0

As it is stated on its web page[9], the SSFNet project has two components: research and development of scalable modeling and simulation tools, and - using these tools - research on the dynamic behavior of very large networks. The software research has been focused on scalability: This includes modeling scalability with number of nodes, traffic flows, bandwidth, system heterogeneity, as well as performance scalability with number of processors. Modeling scalability is a prerequisite for constructing global-scale network models; performance scalability is a prerequisite for their efficient simulation. SSFNet uses and

Figure 4.1: SSFNet logo.

[2] On the digital-media accompanying this report there are examples of the configuration files and logs showing this situation.

engine that can be used to simulate many different things. Also, there are different implementations of the same model. There are C++ and Java implementations, for this work, the one developed over Java was chosen. The main reason was the possibility of running the same copy of the simulator over different machines at the same time. This was a great advantage over a theoretical better performance of a C++ implementation.

History and features

The chosen implementation was the **Raceway SSFnet 2.0**, because it was the most recent one, others can be found on the SSFNet site [9]. Between the most notable features of this simulation suite we can highlight (From [9]):

Scalable high performance Java simulation platforms, often distributed at no cost for research purposes. The best of these are stable, compact, and can execute either serially on a single processor, or in parallel on multiprocessor machines. Parallel execution under Linux, Solaris, and Windows NT using JDK1.2 and higher. Other platforms may support parallelism as well.

A simple, standardized syntax for high-level model description, the Domain Modeling Language (DML). A DML has helped SSFNet users create complex topology Internet models with 100,000 multi-protocol hosts and routers. DML specifies a hierarchy of attributes, with inline attribute substitution and multiple inheritance. DML is simple and easy to read and write directly by modelers, is used in graphical network design and validation tools, and serves as a machine-independent model exchange format suitable for storage in databases.

The latest SSFNet distribution: a collection of DML-configurable Java components for Internet modeling and simulation. The distribution includes source code for:

1. Two derivative frameworks, SSF.OS (for modeling of the host and operating system components, especially protocols) and SSF.Net (for modeling network connectivity, creating node and link configurations).
2. Core Internet protocol models (IP, BGP4, OSPF, TCP, UDP), Sockets, and various workload-generating client and server application models.
3. Protocol validation tests are included. Topological network component addressing and automated IP address allocation (CIDR compliant).
4. Management of global traffic patterns.
5. Management of parallel random number streams, employing a suite of strong random number generators and statistics from the CERN Colt package.

Tutorials explaining step-by-step how to create DML network models , and how to write configurable protocol models using SSF.OS (in x-kernel style).

BGP4 Features : Specifications of: BGP4, RFC-1771[28] (EBGP,IBGP) with all timers. Route Reflection[30], route Flap Damping[16] and no aggregation[28]. It includes a validation suite to test modifications of the implementation.

Opposing to NS-2, this simulator uses a different model. It chooses only one language to implement the simulator (Although this language can change, depending on your needs) but it also uses a special language to define the simulations situations (DML). This language allows to specify a complete network in all its parameters, network layers and events. It uses an object approach, so dictionaries can be defined, allowing to design extremely big networks easily so general network situations can be simulated on an Internet scale.

Also, the BGP4 source code was revised, and it was well structured and highly commented. The presence of the JavaDoc was also considered as it represents an extremely useful tool to get to understand any java-based implementation.

Again, talking about the Java implementation, it was seen as an advantage. The department has 3 Sun machines and we were provided with a remote storage space (Half a Gigabyte) to keep our work. We also had two slower PCs (PIII and PII) to use on the project. So everything was connected (It will be later described) and we wanted to be able to run the same copy of the SSFNet in all the machines, and Java was just perfect for that task.

Set up and usage

If we compare this simulator with Ns-2, this step was much easier. Independently of the machine, it is only needed to uncompress the files and configure some environment variables to make it work. The easiest way is to create a make file that sets the local environment variables and loads the corresponding DML file. An example make file can be found in the appendix.

Testing samples

Again, we tried to create the studied five clique scenario in *fig. 3.1* and compare the simulation output with the theoretical results. We created a DML file with that layout and run the simulation. The results were similar to reality, but still it didn't exactly look like the example. Looking deeper in the log file, something was noticed. When a node would chose other node as the next-hop to a destination, it would send an advertisement to all the neighbors. However, to the one who was going to be next-hop, it would send a withdrawal. This is called split-horizon for BGP4 and it is an active loop detection technique that has a good impact in the behavior (As It will be shown in the next chapter) of BGP4, but to try to validate this simulator we deactivated it. So after deactivating it, we simulated the example again and finally we got a result that looked exactly like the validation example.

So, after this results, the features explained before, and the problems with ns-2 we chose SSFNet as the simulator to keep working on this thesis.

4.2 Simulation Environment

4.2.1 Simulation Parameters and Method

First a simulation method has to be established. It includes a series of elements that will be shown here:

Pool of topologies After looking for topology resources we chose the following models: clique(4-20 nodes), Alternative path model from *fig. 3.2* (10-30 nodes), a bad case for ghost-Flushing, a small case that shows the convergence properties and models from small photographs of the Internet(29, 110, 208, 409 nodes). They show all the situations BGP4 has to deal with: extreme theoretical cases (Good and bad) to examples extracted from the real world. This cases will be detailed and shown in the next chapters.

Simulation Parameters The simulator can be configured to behave closer to reality or closer to the theory. Both types of results are interesting. The theoretical configuration can be used to confirm or deny the theory statements behind the proposed modifications to BGP4. The realistic setting, obviously, is needed before the modifications to be deployed. A real scenario is always harder than the theoretical one. But, how can this be controlled? SSFNet BGP4 implementation provides the following options:

- Random seed/algorithm of every simulation. The seed has to be different in each simulation so we get a good number os samples to be compared.
- The Jitter applied to all the timers used by the BGP4 protocol. Activated when we want a more realistic simulation. Deactivated otherwise.
- Split-Horizon. This technique is used by many real routing softwares (Including Zebra). To do tests showing how the modification proposals would work on existing routing softwares this feature has to be activated.

Applying proposals The versions of BGP4 that are going to be tested are: Traditional BGP4, Ghost Flushing activated, Ghost Buster activated (Over Ghost Flushing). Also the Ghost Buster option is to be used with 3 different delta values used, 20, 30 and 50 seconds. Values under, equal and over the MRAI timer initial value, because the interaction of this two timers (MRAI and delta) seems to be critical to the performance of this proposal.

Compatibility with Traditional BGP4 Some simulations will be also run to test the impact of mixing nodes with ghost-flushing activated and deactivated.

Sampling the results was also important. The rule that we followed is to repeat all the different simulations (We understand one simulation as one topology, with one "reality setting" and one of the versions of BGP4) at least 100 times, to be able to calculate average values. For a single model the average values of every size can be compared. For models with only one side an histogram will be used to show the results.

Simulation Output SSFNet can be configured to generate different outputs about the events happening on the simulations. For the interest of this project, only the messages emission was logged. (We can count the convergence time and number of messages).

Meaning of one simulation The case that we will always analyze is triggered by the sudden failure of a node. We will measure 4 values. Number of messages and time of set up of the simulated environment and Number of messages and convergence time of the system since the failure to the stabilization. This are the values that we will use to compare the performance of the different versions of BGP4. Depending on the topology this values represent the converge on set-up, fail down or fail over situation.

As we can see, there is always a double point of view, looking from the theory and from the real world.

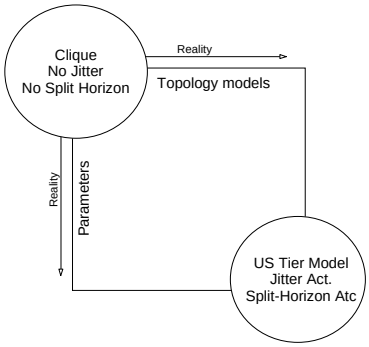

Figure 4.2: Two extremes between reality and theory.

4.2.2 Simulating means programming

Running a high number of simulations can be done in different ways. Since this is an engineering work, it was decided to spend some extra time in developing a small application that would simplify all the functions needed to run a simulation. The main functions to be implemented were:

1. Generating the simulation configuration files. Given a topology (the name and size) and "reality parameters", a the dml file used by the simulation had to be generated.

2. Launching the simulations. A group of simulations were run all together. Specifying the topologies, and different simulation parameters, the dml file generator is used and then the simulator launched.

3. Output processing. The output of each simulation can be large (Hundreds of megabytes) so it has to be automatically processed after it is finished. This means extracting the four values mentioned before: Initial number of messages, initial set-up time, failure number of messages and failure convergence time.

4. Load control, this simulations were run over department machines used by other university personnel. The simulation launcher had to be modified to use more or less cpus from the system (Concurrent simulations) depending of the time of the day. During office hours the load was lower than during nights or weekends.

5. Machine job distribution, the launch software would evaluate which machine was executing it and depending on that, some tasks were executed. This was done so the initial launch of the simulations was machine-independent but the results were different in each machine.

The initial problems form simulating a big number of models (And sometimes individually large) were not always foreseen so, a big part of this software (Specially load control and post-processing) was developed from problems that appeared during this stage.

Software developed

In this section, we will show how the previous tasks were done. Many versions of this software have existed and the evolution comes from problems and new needs that appeared during the simulations. This chapter has the double function of listing the software developed an being a user manual for future users of it.

To develop this software, two scripting languages were chosen, Bourne Shell and Tcl. The first one because of the pipe functionality (really useful to modify and create text files) and TCL because of its powerful programming structure and file management.

Facts to be considered by this software:

Topology files To generate easily simulation models, a small piece of software was written to generate DML files from a simple topology file it has this structure:

```
[Number of nodes]
[Triangular connection matrix]
ghostflushing [true/false]
ghostbuster [true/false]
busterdelta [value of delta timer in seconds]
dead [number of node to die 5] [time of death in second}
```

Concurrency Some code was written to generate some topology files of the models in different sizes. However, some of the models, like the Internet models, are pre-generated DML files, thus, some software was needed to modify this files according to the simulation parameters.

Programs developed

In this section we will show all the software written to make the simulations run. To understand how they interact, the schema on *fig. 4.3* can be consulted.

multiSim.sh

1. Function: General simulation launcher. It runs 2-10 simulations concurrently depending on the time and date.
2. Parameters:
   - Initial Iteration, final iteration: Two natural numbers, the simulation will be repeated $final - initial$ times.
   - BGP versions: Can take the values: NORMAL, GF, GB-D(value) or NADA. It can be used as a list, i.e.: "NORMAL GF". If NADA is specified, the program will assign the version depending on the machine the program runs on: sigma1: NORMAL, sigma2: GF and GB-D20, sigma3: GB-D30, GB-D50.
   - Topology: Can take the values: clique, altpath(Alternate path), white (Small topology), bad (Bad case for ghost-flushing), multi (Models from the internet).
   - Simulation Parameters: Can take the values: sim1(No Random, no jitter), sim2(Random,No jitter), sim3(No Random, Jitter), sim4(Random, Jitter), sim2SH(Random, no jitter, Split-horizon activated), sim4SH(Random, Jitter, Split-horizon activated).
   - Sizes: List of naturals with the sizes of the models. i.e. "5 4 5 6 7 8 9"
3. Execution example:

```
sh multiSim.sh 0 100 "NORMAL GF GB-D20 GB-D30 GB-D50" clique \
        sim2 "4 5 6 7 8 9 10 11"
```

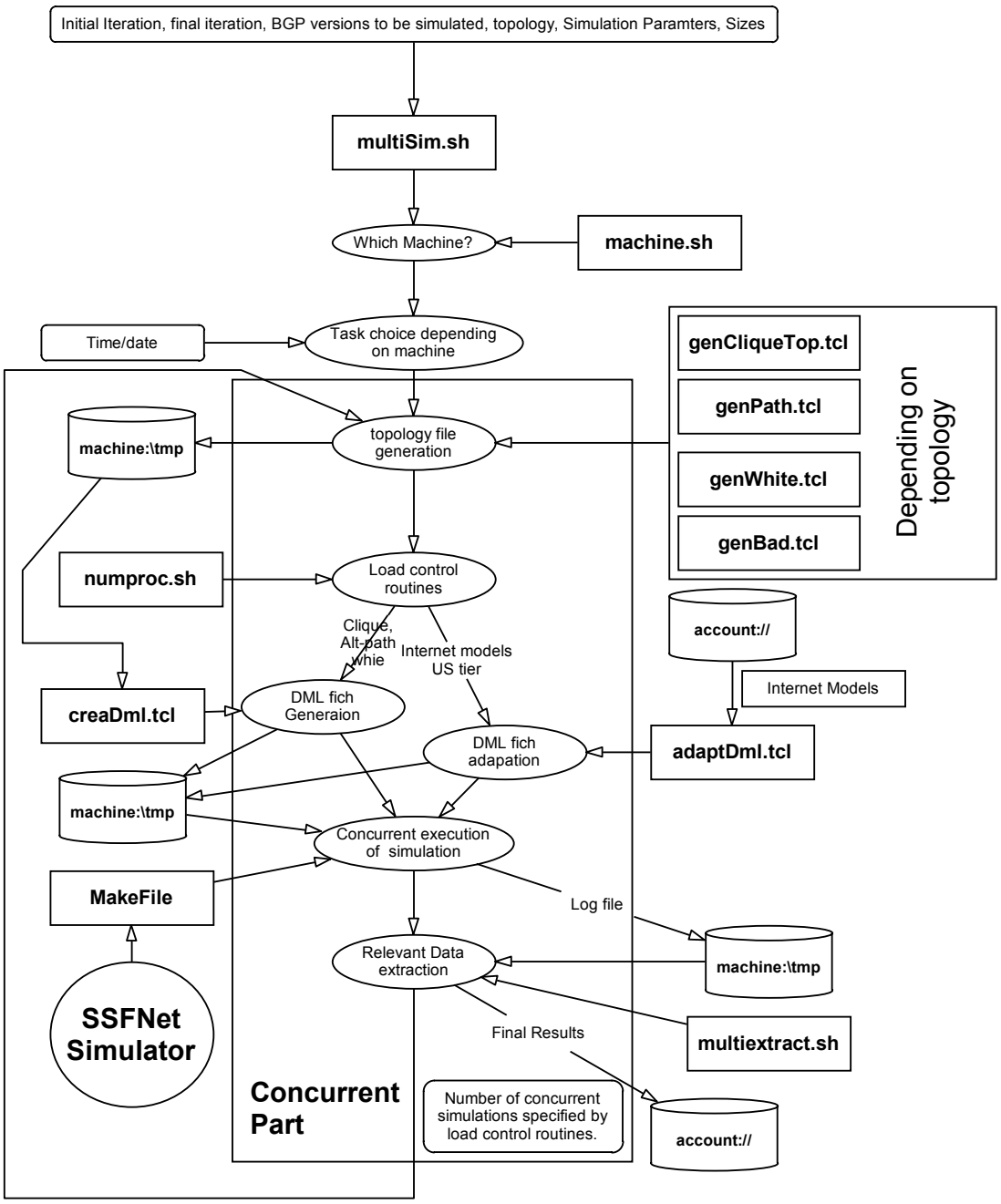

Figure 4.3: Simulation schema.

(It will simulate cliques of sizes 4 to 11 with the traditional BGP4, with ghost-flushing, with ghost-buster delta=20,with ghost-buster delta=30, with ghost-buster delta=50 repeating each case 100 times.)

4. Space used
   - /tmp/gonrod-3/[simulation parameters name]/(topology files).
   - /tmp/gonrod-3/[simulation parameters name]/(DML files)
   - /tmp/gonrod-3/[simulation parameters name]/(simulation log files).
   - ./tmp/[simulation parameters name]/(extracted results).

machine.sh

1. Function: Looks up which machine is running this process.
2. Parameters: None
3. Execution example:

```
sh machine.sh
```

4. Space used: None
5. Response: Machine host-name.

numproc.sh

1. Function: Depending on the date and time it calculates how many simulations there should be at the same time.
2. Parameters: None
3. Execution example:

```
sh numproc.sh
```

4. Space used: None.
5. Response: Number between 1 and 4. In office hours(8-20h) 2, rest of the day and weekends (Sat,Sun) 4.

genCliqueTop.tcl

1. Function: Creation of the topology file of a clique structure with the parameters specified on the command line.
2. Parameters:
   - Clique size: Value: Natural number.
   - Dead node: Can take values os Between 0..Size. If 0, no node dies.
   - Time of death: Simulation time (Seconds) in which the dead node should die.
   - Ghost Flushing: True/false. Activates this rule for all the nodes.
   - Ghost Buster: True/false. Activates this rule for all the nodes.
   - Delta time: Sets the timer of the delta value for the nodes if the Ghost-buster rule is activated.
3. Execution example:

```
tcl genCliqueTop.tcl 5 5 5000 true false 0
```

   (Clique with 5 nodes, node 5 will die at second 5000 of simulation. They run ghost-flushing but not ghost-buster.)
4. Space used: None.
5. Response: On the standard output will write a topology file of the structure before specified.

genPath.tcl

1. Function: Creation of the topology file similar to the one in *fig. 3.2*.
2. Parameters:
   - Network size: Value: Natural number. The model generated will have an alternate path of this value divided (integer division) by two. The rest will be the size of the clique.
   - Time of death: Simulation time (in seconds) in which the dead node should die.The node that dies is the one connecting the alternate path with node 2.
   - Ghost Flushing: True/false. Activates this rule for all the nodes.
   - Ghost Buster: True/false. Activates this rule for all the nodes.

- Delta time: Sets the timer of the delta value for the nodes if the Ghost-buster rule is activated.

3. Execution example:

```
tcl genPath.tcl 11 5000 true true 50
```

(Clique with 11 nodes (altpath of 5 nodes and clique of 6), node will die at second 5000 of simulation. They run ghost-flushing and ghost-buster with a delta timer of 50.)

4. Space used: None.
5. Response: On the standard output will write a topology file of the structure before specified.

genWhite.tcl

1. Function: Creation of the topology file similar to the one in [13], representing a part of the internet.
2. Parameters:
   - Time of death: Simulation time (in seconds) in which the dead node should die.The node that dies is fixed.
   - Ghost Flushing: True/false. Activates this rule for all the nodes.
   - Ghost Buster: True/false. Activates this rule for all the nodes.
   - Delta time: Sets the timer of the delta value for the nodes if the Ghost-buster rule is activated.
3. Execution example:

```
tcl genWhite.tcl 5000 true true 50
```

(Fixed node will die at second 5000 of simulation. They run ghost-flushing and ghost-buster with a delta timer of 50.)

4. Space used: None.
5. Response: On the standard output will write a topology file of the structure before specified.

genBad.tcl

1. Function: Creation of the topology file similar to the one in *fig. 3.5* representing a bad case for ghost-flushing.
2. Parameters:
   - Time of death: Simulation time (in seconds) in which the dead node should die.The node that dies is fixed.
   - Ghost Flushing: True/false. Activates this rule for all the nodes.
   - Ghost Buster: True/false. Activates this rule for all the nodes.
   - Delta time: Sets the timer of the delta value for the nodes if the Ghost-buster rule is activated.
3. Execution example:

```
tcl genBad.tcl 5000 true true 50
```

(Fixed Node will die at second 5000 of simulation. They run ghost-flushing and ghost-buster with a delta timer of 50.)

4. Space used: None.
5. Response: On the standard output will write a topology file of the structure before specified.

creaDML.tcl

1. Function: Converts a topology file(specified before) into a DML file to be used by SSFNet simulator.
2. Parameters:
   - Topology File Name: File name to read the topology information from.
   - Random seed: Random seed used by SSFNet simulator to be included on the DML file.
   - Type of simulation: It can take the values of:sim1(No Random, no jitter), sim2(Random,No jitter), sim3(No Random, Jitter), sim4(Random, Jitter), sim2SH(Random, no jitter, Split-horizon activated), sim4SH(Random, Jitter, Split-horizon activated).
3. Execution example:

   ```
   tcl creaDML.tcl topology.txt lacasito-seed sim2SH > file.dml
   ```

   (DML file with the topology and ghost-rules options on topology.txt. Simulation options: Random activated, jitter deactivated and split-horizon activared)
4. Space used
   - bgpOptions.txt.[simulation parameters name]. Containing DML commands to specify the type of simulation.
5. Response: On the standard output the out dml file will be listed.

adaptDML.tcl

1. Function: Adapts a DML file to use the options specified in the parameters. The death time and the dying node are specified in the original file (It may depend on each case and on the size of the topology)
2. Parameters:
   - Topology File Name: File name to read partial DML file.
   - Random seed: Random seed used by SSFNet simulator to be included on the DML file.
   - Type of simulation: It can take the values of:sim1(No Random, no jitter), sim2(Random,No jitter), sim3(No Random, Jitter), sim4(Random, Jitter), sim2SH(Random, no jitter, Split-horizon activated), sim4SH(Random, Jitter, Split-horizon activated).
   - Ghost Flushing: True/false. Activates this rule for all the nodes.
   - Ghost Buster: True/false. Activates this rule for all the nodes.
   - Delta time: Sets the timer of the delta value for the nodes if the Ghost-buster rule is activated.
3. Execution example:

   ```
   tcl adaptDML.tcl multi29.dml lacasito-seed sim2SH false false 0> file.dml
   ```

   (DML file with the topology on multi29.txt. Simulation options: Random activated, jitter deactivated and split-horizon activared). All nodes will run traditional BPG.)
4. Space used
   - bgpOptions.txt.[simulation parameters name]. Containing DML commands to specify the type of simulation.
5. Response: On the standard output the dml file will be listed.

multiextract.sh

1. Function: Analyzes the result in a simulation log file. It generates the values: Set-up time, set-up messages, failure convergence time and failure number of messages. It removes the log file after.
2. Parameters:
   - Log file name.

- Output file name: It generates 4 files: name.initial, name.initialmessages, name.final, name.finalmessages corresponding to the four values calculated.

3. Execution example:

```
sh multiextract.sh a.log output.txt
```

4. Space used: It uses the program buscafail.tcl, this program searches in a log file for all the values needed.

multimerge.sh

1. Function: Puts together all the data generated by the simulation in only one file.
2. Parameters:
   - Name of the output.
   - Simulation name.
   - Topology.
   - Versions of the protocols.
3. Execution example:

```
sh multimerge.sh report sim2SH clique "NORMAL GF"
```

4. Space used: It reads the files in ./tmp/[simulation]

Makefile

1. Function: Run the simulation file, creating a output file, during the running time and using a memory limit specified to it.
2. Parameters:
   - DMLFICH, Name of the origin DML fich.
   - NOMFICH, Name of the output log file.
   - RUNTIME, number of seconds to run the simulation (In simulation time).
   - MAXMEM, number of memory megabytes at most to be used.
3. Execution example:

```
make DMLFICH=a.dml NOMFICH=/tmp/log.txt RUNTIME=10000
MAXMEM=4000
```

(It will run the SSFNet simulator with the a.dml file. The output will be written to /temp/log.txt. At Second 10000 of simulation, it will stop, even if there are events to happen and the Java machine only will be able to use 4Gb of memory.)

4.2.3 The Hardware

A factor that made us decide for SSFNet was the hardware environment. To our disposition were two personal computers (Full disposition) and three department machines (Really careful disposition). All of them mounting a remote file system where the simulator and results were stored. The department machines were faster and had much more memory so we chose these ones to run our simulations on. They were three machines with the same features, here are the some of the specs:

Names sigma1.sm.luth.se, sigma2.sm.luth.se, sigma3.sm.luth.se

Model name Sun Fire V480 Server[10].

CPU Four CPUs: 1.2 GHz UltraSPARC III Cu

RAM Memory 16 Gigabytes

Operating System Sun Solaris 9

This simulations were stored in a department store server named claudius.sm.luth.se. All the machines had the same account mounted so they all stored their results in the same space, while they used their own /tmp directory to store the enormous temporary log files generated by the simulations. In *fig. 4.4* we can see a schema of the system. Also in *fig. 4.5* we can see a actual photo of the servers.

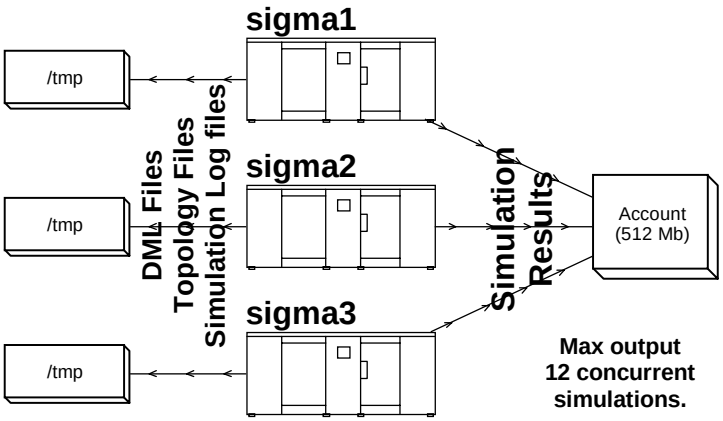

Figure 4.4: Interaction between Sigmas and a local account.

Figure 4.5: LTU's Dator Hall, the Sigmas are the purple machines in the middle-top. Photo by courtesy of Johan Hallbäck.

4.3 Analysis of SSFNet's BGP4 Implementation

In this section we'll make an analysis of relevant parts of the BGP module implementation. This analysis intends to be a guideline to future modifications to this module.

4.3.1 Source Structure

All the code concerning the BGP4 can be seen in *fig. 4.6*

~/src/SSF/OS/BGP4:
- LocRIB.java: Abstract class implementing RIB data structure.
- AdjRIBIn.java
- AdjRIBOut.java
- RIBElement.java
- DampInfo.java
- InBuffer.java
- Monitor.java
- Route.java: Representation of route as it´s received.
- RouteInfo.java: All information about a route that BGP uses.
- RouteInfoOOC.java
- RouteInfoIC.java
- Debug.java
- PeerEntry.java: Class representing the data used by a BGP instance referring to a BGP-Peer.
- WeightedInBuffer.java
- **BGPSession.java**: Main class Implementing the BGP4 protocol algorithm.
- Global.java: BGP Global configuration options.

~/src/SSF/OS/BGP4/Comm/: BGP Messages Classes.
~/src/SSF/OS/BGP4/Path/: Path Attributes Classes.
~/src/SSF/OS/BGP4/Players/: Binary Data Treatment.
~/src/SSF/OS/BGP4/Policy/: BGP Policy Implementation.
~/src/SSF/OS/BGP4/Timing/: BGP timers and timed events.
~/src/SSF/OS/BGP4/Util/: Internal Data Representation.
~/src/SSF/OS/BGP4/Widgets/: Classes Used to implement simulation scenarios. (Session killers)
~/src/SSF/OS/BGP4/doc/: Documentation.
~/src/SSF/OS/BGP4/test/: Pre-generated simulation scenarios used to verify the correct behaviour of the simulator.

Figure 4.6: SSFNet's BGP4 code structure.

The behavior of the protocol is implemented in the BGPSession class, that means that, for the Ghost Flushing rule, all the modifications needed were made over this class. So now we are going to show the structure of this class

4.3.2 BGPSession Class

This class implements the core of the BGP4, so first we are going to list the most relevant methods (Explaining their function) and after, we'll show the structure of the most important ones concerning our modifications.

In this chapter we are doing a simplified schema of the code. We have eliminated some parts (Not relevant to the modifications) to make easier the understanding of the code

Methods

void config(com.renesys.raceway.DML.Configuration cfg) Sets configurable values for BGP. Here we can add the code needed for the new configuration options related to the GhostFlushing/Buste rules.

void decision_process_1(java.util.ArrayList infolist) Runs Phase 1 of the Decision Process, which is responsible for calculating the degree of preference of newly added or updated routes.

java.util.ArrayList decision_process_2(java.util.ArrayList changedroutes, boolean dampReuse) Runs Phase 2 of the Decision Process, which is responsible for selecting which routes (from Adj-RIBs-In) should be installed in Loc-RIB.

void decision_process_3(java.util.ArrayList locribchanges) Runs Phase 3 of the Decision Process, which is responsible for disseminating routes to peers.

int dop(Route rte) Calculates the degree of preference for a route.

void external_update(java.util.HashMap wds_table, java.util.HashMap ads_table) Tries to send update messages to each external peer if there is any new route information in Adj-RIBs-Out to be shared with them.

void force_send(Message msg, PeerEntry peer) (Overloaded) Sends a message immediately without incurring any CPU delay.

boolean handle_event() This process handles both externally and internally generated BGP events.

void handle_mrai_exp(TimeoutMessage tmsg, PeerEntry peer) Handles an MRAI Timer expiration.

void handle_update(UpdateMessage msg) This method takes all necessary action when an update message is received.

boolean push(ProtocolMessage message, ProtocolSession fromSession) This process optionally imposes a processing delay for certain BGP events, then passes them on to the receive method to be handled.

void send(Message msg, PeerEntry peer) (Overloaded) Generic procedure to take any kind of BGP message and push it onto the protocols below in the stack.

void try_send_update(UpdateMessage msg, java.util.ArrayList senders, PeerEntry peer) Handles the sending of an update message. This method takes into account the expiration of MRAI timer and management (Adding Routes) of the waiting list associated with that timer.

decision_process_2

1. Function: Takes all necessary action when an update message is received.
2. When: an update message arrives.
3. Data Input: Update message that just arrived
4. PseudoCode:

```
peer=peer from which we received the message
rcvd_rtes = routes extracted from the UpdateMessage
rcvd_wds= nlris to be withdrawn extracted from the UpdateMessage.
Boolean rundp=false, indicates if decision process must be run
Check feasibility of rcvd_rtes.
Check AS path loops in rcvd_rtes.
Changedinfo -> array of routes info.
Stores withdrawn and replaced routes.
newinfo_list -> array of routes info.
Stores new routes.
For (each nlri in rcvd_wds) loop
   Create routeinfo with withdrawn nlri.
   Mark routeinfo as not feasible.
   Put routeinfo in changedinfo.
   If (there is a route about nlri in the RIBIn of peer) then
       Remove route from RIBIn.
```

```
        Mark removed route as not feasible.
        ChangedInfo.
        add(removed route)
        Rundp = true
        Remove this route from all the waiting lists (to be sent because
        of mrai limitation) associated with al peers.
    End if
End loop

For (each route in rcvd_rtes) loop
    Rundp=true
    Put route in RIBIn.
    Oldinfo=replaced route(if any) from RIBIn. Newinfo_list.add(route)
    If (oldinfo!=null) then
        Mark oldinfo as: not feasible, implicitly withdrawn.
        Mark newinfo as: implicit withdrawal.
        Changedinfo.add(oldinfo)
    End if
End loop
If (rundp) then
     Decision_process_1(newinfo_list)
     Changes_in_loc_RIB=Decision_process_2(changedinfo)
 Decision_process_3(Changes_in_loc_RIB)
End if
```

try_send_update

1. Function: Tries to send a message. If it's not possible (protocol limitation) it takes the proper actions..
2. When: when ever a message is needed to be sent.
3. Data Input: UpdateMessage to be sent, the address of the advertisers of the route and to which peer we want to send the message.
4. PseudoCode:

```
msg = UpdateMessage to sent.
Senders = Addresses of the original advertisers of each route on the message.
Peer = peer to whom we want to send the message.
[Code initialising Timers when this method is run for the first time]
if (MRAI not expired yet) then
    Eliminate withdrawals about nlri that is referenced by the routes on the
     updatemessage: Implicit withdrawal.
    Update Waiting lists.
    If a route is going to be withdrawn eliminate it from the waiting lists.
    Put the routes from the message in the waiting list (MRAI is not expired yet)
    Eliminate those routes from the UpdateMessage.
    Send UpdateMessage.
    Return()
end if
Eliminate withdrawals about nlri that is refered by the routes on the
 updatemessage: Implicit withdrawal.
Update Waiting lists. If a route is going to be withdrawn eliminate it from
the waiting lists.
Send UpdateMessage.
```

4.3.3 The Update Process

After showing the internals of the main methods that take part in the BGP protocol behavior we are going to try to show how they work together from the moment and Update Message is received to the moment new routes or withdrawals are widespread. The schema can be found in *fig. 4.7*

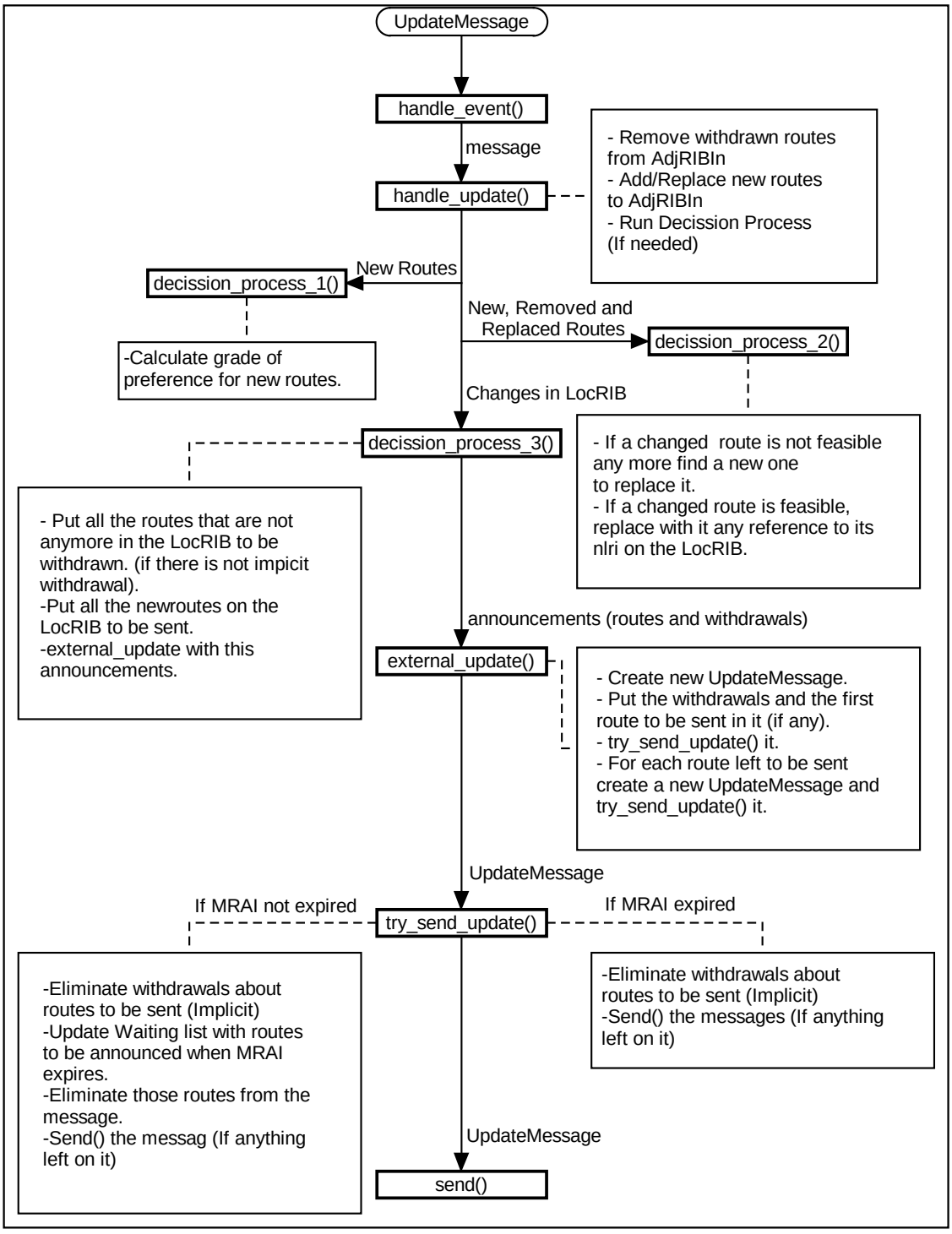
UpdateMessage
handle_event()
message
handle_update()
- Remove withdrawn routes from AdjRIBIn
- Add/Replace new routes to AdjRIBIn
- Run Decission Process (If needed)
New Routes
decission_process_1()
-Calculate grade of preference for new routes.
New, Removed and Replaced Routes
decission_process_2()
Changes in LocRIB
decission_process_3()
- If a changed route is not feasible any more find a new one to replace it.
- If a changed route is feasible, replace with it any reference to its nlri on the LocRIB.
- Put all the routes that are not anymore in the LocRIB to be withdrawn. (if there is not impicit withdrawal).
-Put all the newroutes on the LocRIB to be sent.
-external_update with this announcements.
announcements (routes and withdrawals)
external_update()
- Create new UpdateMessage.
- Put the withdrawals and the first route to be sent in it (if any).
- try_send_update() it.
- For each route left to be sent create a new UpdateMessage and try_send_update() it.
UpdateMessage
If MRAI not expired
try_send_update()
If MRAI expired
-Eliminate withdrawals about routes to be sent (Implicit)
-Update Waiting list with routes to be announced when MRAI expires.
-Eliminate those routes from the message.
-Send() the messag (If anything left on it)
-Eliminate withdrawals about routes to be sent (Implicit)
-Send() the messages (If anything left on it)
UpdateMessage
send()

Figure 4.7: Update Process in SSFNet's BGP Schema.

4.4 Modifications to SSFNet's BGP4 for the Ghost-Flushing Rule

To implement this rule inside the simulator we had to take a deep look at the Update Process (From the UpdateMessage arrival to the generation of a new one). We found that, to make the protocol to behave according to the Ghost Flushing rule, we had to implement two tasks: Identifying the "Ghost Information" and Sending the new withdrawals when needed.

To **identify the Ghost Information** we decided that the best point in the BGP code was inside the second phase of the decision process (decision_process_2()). In that moment, we have access to the LocRIB (Route we were using until now) and the new route that is going to be put inside the LocRIB (and which can be ghost information). If we find that and old route is going to be replaced by a worse route, we can say that, according to the ghost flushing rule, there is ghost information. To keep account of the ghost information we also use a data structure storing the nlris that we received ghost information about. Associated to it, also, last ASPath length that we have put on the LocRIB (there can be many replacements in a short time) and the length of the ASPath of the route that was on the LocRIB before any replacement. Using this, we can decide later to send or not withdrawals according to the rule. (The last replacement was kept because there can be more than one replacement on the locRib before the decision process phase 3.We have to take into account the length from the old route before any replacement and the length of the route that is in the LocRIB after all the decision process).

The place to insert the **new withdrawals sending** was easy. We decided the best place was the third phase of the decision process (decision_process_3()). In that moment the protocol decides how to spread the routes and withdrawals, so we had just to modify a little bit. If the old route was shorter than the one now on the LocRIB we send a withdrawal to all peers about the NLRI of those routes.

4.4.1 Activating The Ghost-Flushing feature from the DML file

The configuration file reading system was also modified. Two configuration environments were modified, the global configuration environment and the local configuration environment. The local assertions are used when we are generating the models from scratch (So we generate the parts for each BGP4 node). The global environment assertions will affect all the nodes and they are useful when working with pregenerated DML files. In those cases modifying all the nodes can be long.Assertions defined:

bgpoptions environment Overrides local setting.

- forceghostflushing true/false, sets the usage of this rule.
- ghostdebug true/false, sets the debugging of messages of the ghost rule.

BGPSession environment

- ghostflushing true/false, sets the usage of this rule.

4.4.2 Modification Schema

In *fig. 4.8* we can find a reduced schema of the Update Message schema with the modifications in bold letters.

4.4.3 Deeper Inside the Modification (Pseudocode)

We are going to show how this modifications impact on the pseudocode described in previous chapters. We had to modify to methods: decision_process_2 and decision_process_3. (Modifications in bold letters).

GhostData Structure

1. General Description: This Data structure holds the Ghost Information list that has been found during the decision_process_2. It is initialised at the beginning of the decision process. Currently it's formed by an array of the type GhostNLRI. This type stores: nlri from the GhostInfo, the length of the old path and the length of the path that was put the last on the loc rib. In the future this GhostData structure will be a hash map indexed by the nlri.

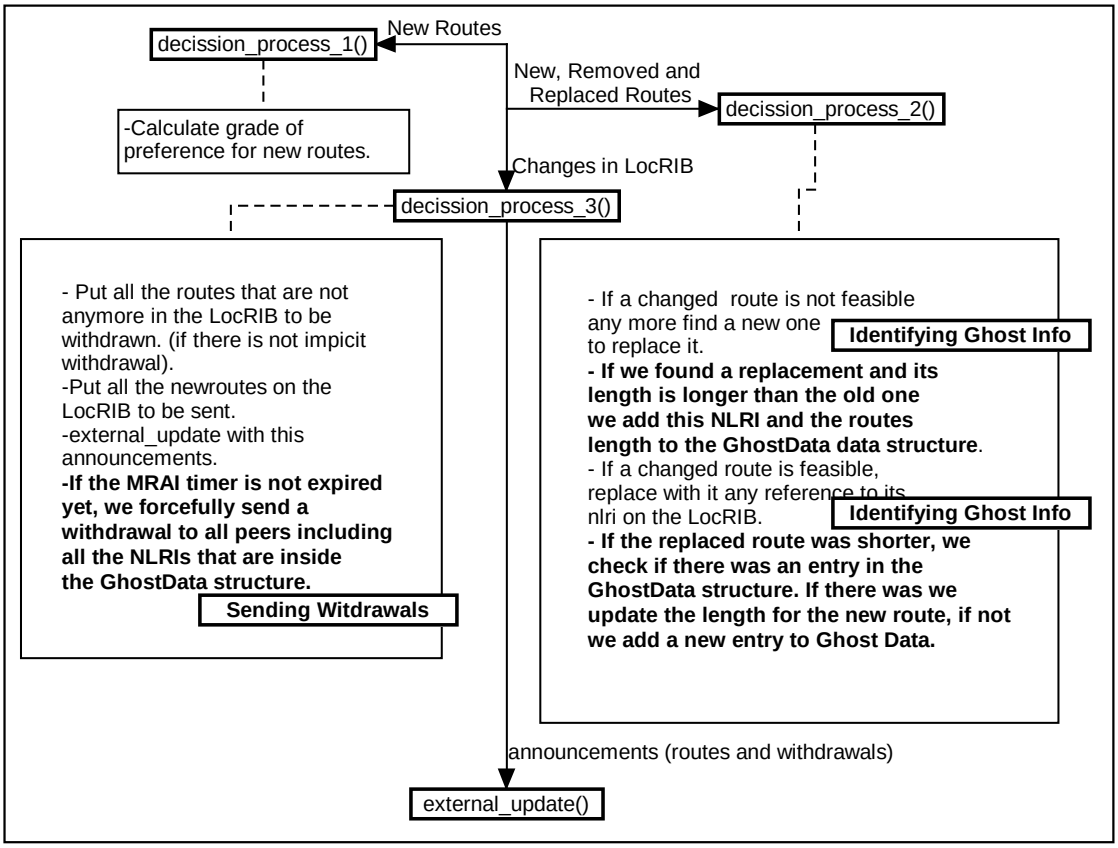
decission_process_1()
New Routes
New, Removed and
Replaced Routes
decission_process_2()
-Calculate grade of
preference for new routes.
Changes in LocRIB
decission_process_3()
- Put all the routes that are not
anymore in the LocRIB to be
withdrawn. (if there is not impicit
withdrawal).
-Put all the newroutes on the
LocRIB to be sent.
-external_update with this
announcements.
-If the MRAI timer is not expired
yet, we forcefully send a
withdrawal to all peers including
all the NLRIs that are inside
the GhostData structure.
Sending Witdrawals
- If a changed route is not feasible
any more find a new one
to replace it.
Identifying Ghost Info
- If we found a replacement and its
length is longer than the old one
we add this NLRI and the routes
length to the GhostData data structure.
- If a changed route is feasible,
replace with it any reference to its
nlri on the LocRIB.
Identifying Ghost Info
- If the replaced route was shorter, we
check if there was an entry in the
GhostData structure. If there was we
update the length for the new route, if not
we add a new entry to Ghost Data.
announcements (routes and withdrawals)
external_update()

Figure 4.8: Ghost Flushed Update Process in SSFNet's BGP Schema.

2. Code for the GhostNLRI type:

```
class GhostNLRI {
    // Domain the ghost info is about.
    public IPaddress NLRI;
    // Length of the first AS path for the NLRI.
    public int oldPath;
    // Length of the longest AS path fot the NLRI.
    public int longPath;
    // Length of the path that actually will remain on the loc_rin
    public int lastPath;

    // Method GhostNLRI
    // It creates an instance of the Class.
    public GhostNLRI(IPaddress IP, int length) {
        NLRI = IP;
        oldPath = length;
        longPath = length;
        lastPath = length;
    }

    // Method NewPath
    // It takes in consideration that a new path about this NLRI is being treated in
    // the update process (Decission process).
    // It returns if the lastPath would be worse thant the first one.
    public boolean newPath(int length)
```

```
    {
        lastPath = length;
        if (lastPath > longPath) {
            longPath = lastPath;
        }
        return (isGhost());
    }
    public boolean isGhost()
    {
        return (lastPath > oldPath);
    }
}
```

decision_process_2()

1. Function: takes the result from the first phase of the decision process and chooses which routes must be installed on the LocRIB. **Also identifies ghost information that can have been installed on the LOCRib.**
2. When: After handling an update if the decision process is needed.
3. Data Input: All the routes that have changed: New Routes, Withdrawn routes and routes that are replaced (Implicit withdrawal).
4. Pseudocode

```
for (every changed Route (New, Replaced and Withdrawn)) loop
    if (route is not feasible) then
        if (route was inside the LocRIB) then
            Remove route from LocRIB.
            Try find a new route for the removed nlri in the RIBIns from other
             neighbours and put it in
            the LocRIB.
            //GHOST-CODE
            If (new Route is longer than the old route) then
                GhostData.add(NLRI, old_path_length, new_path_length).
            End if
            //GHOST-CODE-END
        end if
    else-route is feasible
        if (there is no route in the LocRIB for this NLRI ) then
            Add route to the LocRIB
        Else if (route is better than the one in LocRIB) then
            Remove the current route in LocRIB for this NLRI (if it exists).
            Check if what we replaced was already a replacement (because of a
             withdrawal) and if it was the
            withdrawal that caused the previous replacement does exist.
            Put route in the LocRIB
            We check if there is an entry in GhostData for this NLRI.
            //GHOST-CODE
            If (there is an entry in GhostData) then
                We update the length of the last path about this NLRI in GhostData:
                GhostData.modify(NLRI, new_path_length).
            else
                If (new Route is longer than the old route) then
                    GhostData.add(NLRI, old_path_length, new_path_length).
                End if
            End if
```

```
            //GHOST-CODE-END
        end if
    end if
end loop
```

decision_process_3

- Function: Disseminates routes to peers (Routes and withdrawals). It creates a message with the new routes and the new withdrawals (if any) and it is send at the end (If there is something to send). **It also sends the withdrawals associated with the Ghost-Flushing Rule.**
- When: After handling an update if the decision process is needed.
- Data Input: Changes to the LocRib, GhostData
- Pseudocode:

```
for (each BGP-peer-neighbor) loop
    create wds2send -> List of withdrawals to sent to this peer.
    for (each route changed in LocRIB) loop
        if (route is no in LocRIB anymore) then
            Remove it from RIBOut
            Wds2send.add(route.nlri)
        end if
    end loop
end loop

for (each BGP-peer-neighbor) loop
    create ads2send -> List of advertisements to sent to this peer.
    get wds2send for this peer (created in the previous loop)
    for (each route changed in LocRIB) loop
        if (route is now in LocRIB) then
            Replace any info in RIBOut related to route.nlri with route.
            ads2send.add(route)
            If (route.nlri is in wds2send) then removenlri from wds2send (implicit withdrawal).
        end if
    end loop
end loop
//GHOST-CODE
if (GhostData is not empty) then
    create UpdateMessage.
    for (each entry in GhostData) loop
        if (old_path_length > last_path_length) then
            add nlri to withdrawals in message.
        end if
    end loop
    if (withdrawals in message is not empty)
        force send UpdateMessage to all peers.
    end if
end if
//GHOST-CODE-END

 Check if all the peers are still connected.
 Execute "external_update" with the list of withdrawals and advertisements generated.
```

4.5 Modifications to SSFNet's BGP4 for the Ghost-Buster Rule

4.5.1 Activating The Ghost-Buster feature from the DML file

The configuration file reading system was also modified. Two configuration environments were modified, the global configuration environment and the local configuration environment. Assertions defined:

bgpoptions environment Overrides local setting.

- forceghostbuster true/false, forces the usage of this rule by all nodes.
- ghostdebug true/false, sets the debugging of messages of the ghost rule.

BGPSession environment

- ghostbuster true/false, sets the usage of this rule.

4.5.2 The main idea for the modification

In this case, modifying the protocol to make it comply with the Ghost Buster rule wasn't as easy as for the Ghost Flushing rule. Following the strategy to implement the previous rule we tried to identify the main tasks needed for the Ghost Buster rule. We found four main tasks: Creation and initialization of the Delta timer associated to a route (when it is received by the protocol), modification of the sending method so we respect the delta, handling of the event delta timer expiration. Also it was needed to implement timers, messages and other classes needed to make all this work.

For the **creation and initialization of the delta timer** associated to a route, It was thought that the best place to implement it was inside the code of the AdjRIBIn class. We modified the code of that class so every time a route was added to an AdjRIBIn a Delta Timer was created and associated to the route. It works the same way if the route was eliminated from the AdjRIBIn, the timer associated was eliminated.

About **the sending method**. We altered the method try_send_update so, if the delta timer for this route is not expired yet, it won't be sent or put inside the waiting lists. The part that manages the expiration of the delta timer will do it later.

Handling the expiration of the delta timer, for this, we created a new event inside the BGP4. The idea is quite simple, we added an entry inside the method handle_event that manages every route that has its delta timer expired. If the route is in the LocRIB, it uses the try_send_update method but with the info about the route whose delta timer just expired. It tries to send it according to the classical limitations of BGP4 (MRAI, etc) like delta timer never existed.

Messages, Timers and video tapes. To make all this work we needed some new classes, including a new type of timer that triggered the appropriated event inside the FSM: **DeltaTimer**. It is created and associated to a route when the route arrives. When it expires produces a message that we had also to write (Another class): **DeltaTimeoutMessage**. This message carries a reference to the route triggering the event, so the handle_event knows what to do. To keep in a data structure all the timers together with the cross references between them and their routes we created the **BusterRoute** class. It helps managing the creation and manipulation of the timers. Finally, to index all the BusterRoute instances in the class representing every peer there is a instance of the **ListBusterRoute** class. It consists of a hash table indexed by the NLRI of the route and some methods to make our life a little bit easier.

4.5.3 Making this complex schema work

Now we are going to try to show how is the life of a route since its received until it is sent to other peers, the new route "lifecycle". Considering the complexity of this modification we'll only include what is different from the Ghost Flushing rule.

Route arrives and the protocol tries to send if but delta timer is not expired.

This schema can be found in *fig. 4.9*

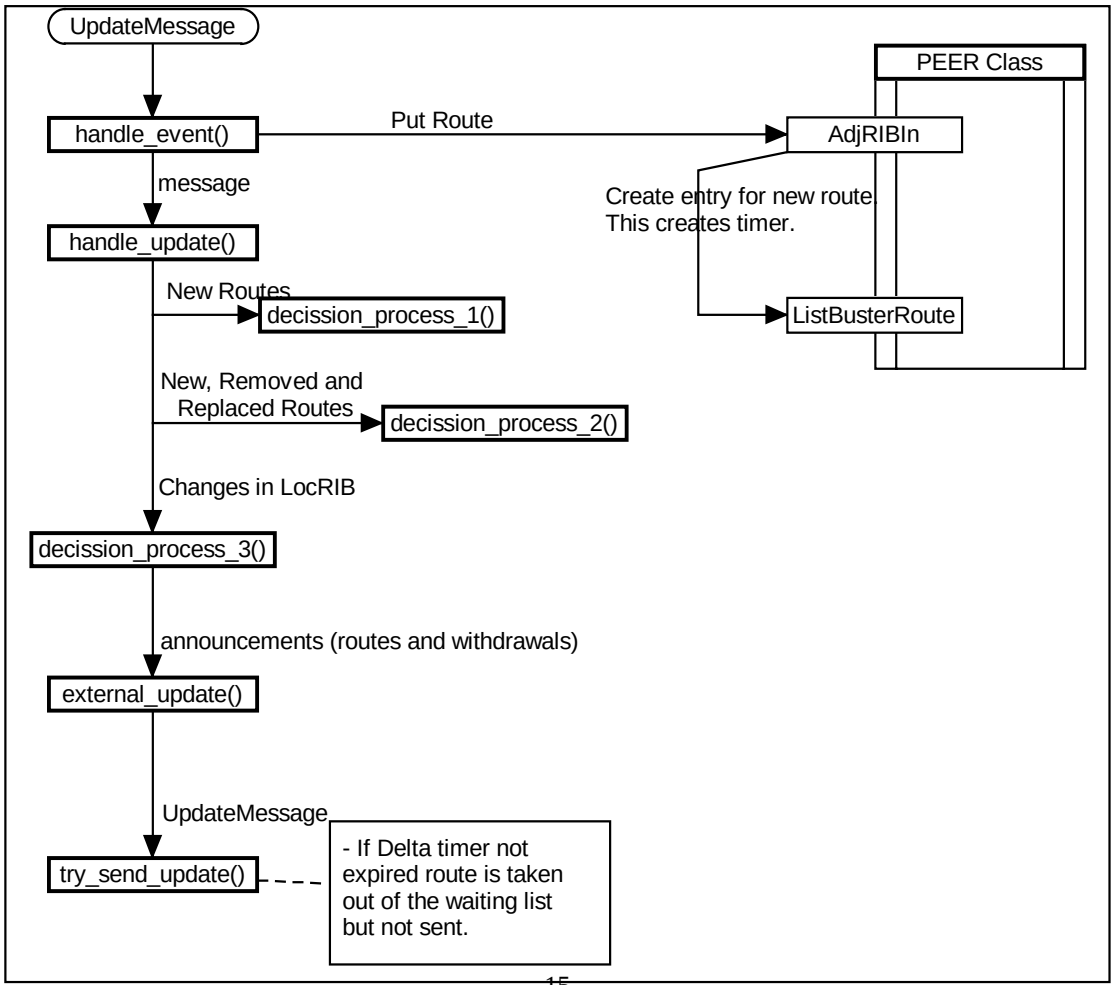

Figure 4.9: Route arrives and the protocol tries to send if but delta timer is not expired.

Delta Timer Expires

We suppose that the route is in the LocRIB (So we want to sent it). The schema can be found in *fig. 4.10*

4.5.4 Deep inside the code (Pseudocode)

We are going to show some modifications to the code on the BGPSession class. Other classes have been also modified (RIBIn, Peer) but they are minor modifications that don't need to be shown. Modifications highlighted with bold letters.

try_send_update()

1. Function: Tries to send a message. If it's not possible (protocol limitation) it takes the proper actions. **Now it also considers the delta timer limitation associated to each route that is going to be sent.**

2. When: when ever a message is needed to be sent.

3. Data Input: UpdateMessage to be sent, the address of the advertisers of the route and to which peer we want to send the message.

4. PseudoCode:

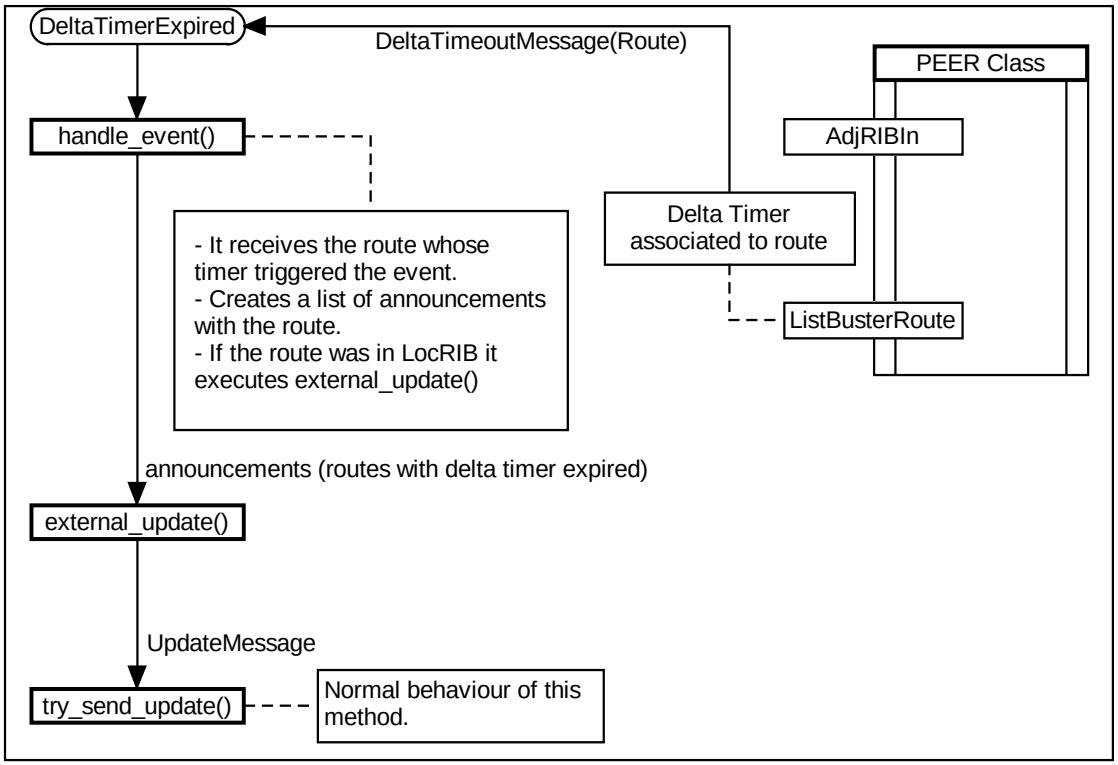

Figure 4.10: Delta Timer Expires.

```
 msg = UpdateMessage to sent.
 Senders = Addresses of the original advertisers of each route on the message.
 Peer = peer to whom we want to send the message.
//GHOST-CODE
 if (route to be sent in message has not its delta timer expired) then
    extract route from UpdateMessage.
 end if
//GHOST-CODE-END
 [Code initializing Timers when this method is run for the first time]
 if (MRAI not expired yet) then
    Eliminate withdrawals about nlri that is refernced by the routes on the
     updatemessage: Implicit withdrawal.
    Update Waiting lists. If a route is going to be withdrawn eliminate it
     from the waiting lists.
    Put the routes from the message in the waiting list (MRAI is not expired yet)
    Eliminate those routes from the UpdateMessage.
    Send UpdateMessage.
    Return()
 end if


 Eliminate withdrawals about nlri that is refered by the routes on the
  updatemessage: Implicit withdrawal.
 Update Waiting lists. If a route is going to be withdrawn eliminate it from
  the waiting lists.
 Send UpdateMessage.
```

handle_event ()

1. Function: Handle both external and internal events. Modelates the BGP4 FSM.
2. When: Always an event arrives or is triggered inside the BGP.
3. Data Input: Event Happened
4. PseudoCode:

```
... old code ...
 In state ESTABLISHED

Case event is Expiration of a Delta Timer:
    If (route associated to timer is in LocRIB) then
        Create announcements for all the neighbor about the route.
        Send those announcements using external_update()
    End if
... old code ...
```

4.5.5 Data structures needed to make all this work

- **DeltaTimer Class**

  General Description This is a timer initialized with a Delta value (seconds). When it expires it sends a DeltaTimeoutMessage message on the FSM of the BPG4 (using method push).

  Location ~/src/SSF/OS/BGP4/Timing/DeltaTimer.java

- **DeltaTimeoutMessage class**

  General Description it is the message generated by the DeltaTimer. It contains the nlri of the route that triggers this message and a reference to the peer where the route is stored.

  Location ~/src/SSF/OS/BGP4/Timing/DeltaTimeoutMessage.java

- **BusterRoute class**

  General Description this class interconnects the route and the Timer. It has methods helping creating the timer and managing it.

  Location ~/src/SSF/OS/BGP4/ListBusterRoutes.java

- **ListBusterRoutes class**

  General Description Holds all the BusterRoute instances associated to the routes of a peer. In deep it's composed by a hash table (hashed by the nlri) and some methods helping to manage the BusterRoutes

  Location ~/src/SSF/OS/BGP4/ListBusterRoutes.java

4.5.6 Class Schema

This schema pretends to show the interconnection between the classes mentioned. It just reflects how the classes point each other.

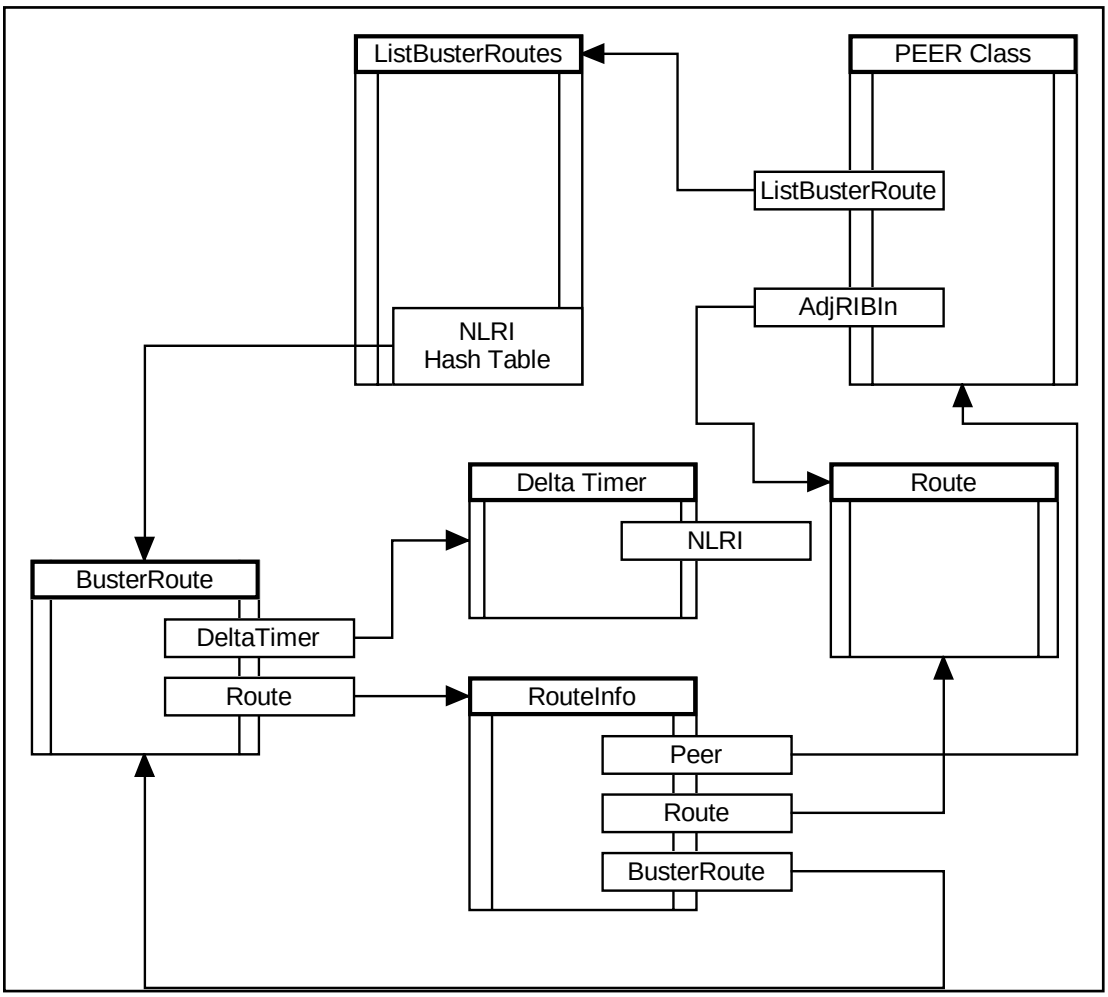

Figure 4.11: Ghost Buster modification, Classes interaction.

Chapter 5

BGP4 Cases Simulation

5.1 Simulated Cases

In this chapter we will show each of the topologies used to analyze the performance of the BGP4 and the rules proposed to modify it.

5.1.1 Clique

Clique topology is the worst case scenario in the convergence problem that we are studying. The more the nodes are interconnected, the more ghost information that will be created. This topology has a big theoretical interest as a worst case scenario. The tests done are just about how it sets up and how it reacts to the failure of one of the nodes. The node that goes down is one of the nodes of the alternate path that connects to the clique.

Sizes from 4 to 20 nodes.

Convergence Times studied E_{down} and E_{up}

Interest Theoretical.

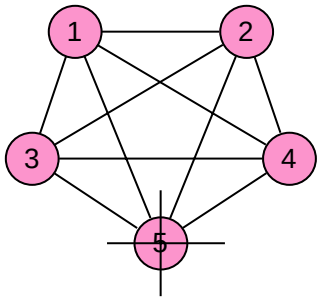

Figure 5.1: Full connected 5 nodes clique.

5.1.2 Alternate Path (AltPath)

This topology is meant to test the capabilities of Ghost Flushing and Ghost Buster in a Fail over situation. This topology complies with this property: Given n as the total number of nodes, there is an alternate path of size $\frac{n}{2}$ connecting two nodes of a clique of size $n - \frac{n}{2}$. It has a big theoretical interest, however such situations appear on the internet, parts of the network highly interconnected that also have a long alternate path (Nomad paths[12]) connecting two nodes.

Sizes from 10 to 30 nodes.

Convergence Times studied E_{over} and E_{up}

Interest Theoretical and slightly practical .

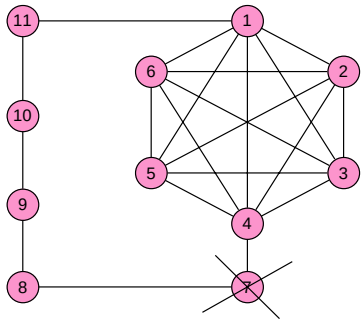

Figure 5.2: Alternate Path scenario of 11 nodes. The alternate path has 5 nodes ($\frac{n}{2}$) and the clique 6 ($n - \frac{n}{2}$).

5.1.3 "White" model

This topology is taken from [13]. It is a simplified model of a network configuration found on the internet. It is more practical than previous examples, but still simplified so it's still slightly theoretical. It is also similar to a topology found on [12]. Since we don't have similar models to this one from different sizes the data will be shown in an histogram style.

Size 8 nodes.

Convergence Times studied E_{down} and E_{up}

Interest Practical and slightly theoretical.

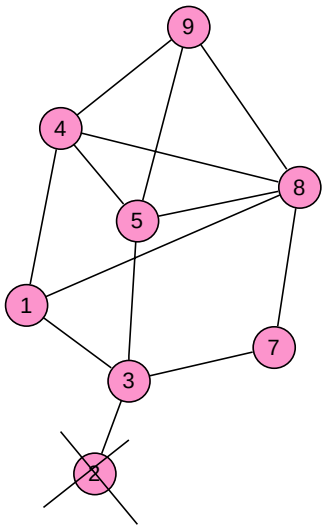

Figure 5.3: "White topology".

5.1.4 "Bad" model

This topology is the bad extreme case for the ghost-flushing rule. The idea of simulating this case is to see how bad is actually compared with the traditional BGP4. It was also extracted from [13]. As in the "white" model, we will show the data in a histogram style. The property a topology has to comply to be similar to this is:

- There is a node A that provides to a node B connectivity to a destination D

- From node A there are many paths to D.
- There has to be at least three paths: P_1, P_2, P_3
- P_1, P_2 share a part of their ASPaths.
- P_3 must not include the common part of P_1, P_2.
- P_3 must be longer.
- The fault must happen in the common part of P_1, P_2.

Size 12 nodes.

Convergence Times studied E_{over} and E_{up}

Interest Practical and slightly theoretical.

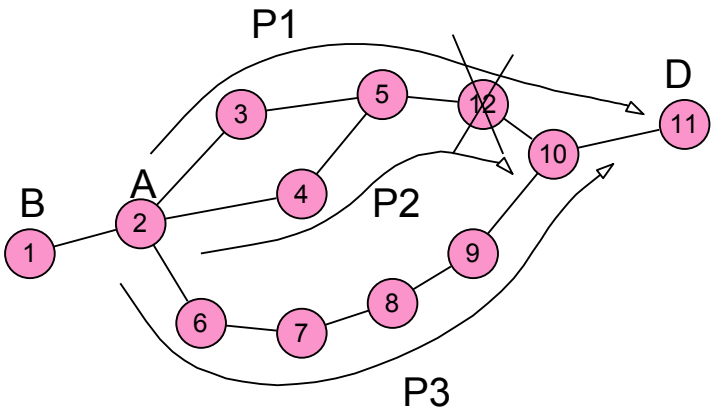

Figure 5.4: "Bad topology'.

The reason that makes this a bad case is when node 12 goes down, Node 2 changes from P_1 to P_2 and advertises it to 1. The Node realizes that P_2 is not valid either and then chooses P_3, longer than P_2, that triggers the ghost-flushing, so it sends a withdrawal about D to node 1, so node 1 doesn't have a path to D. Until MRAI time(30 seconds) later it won't be able to send the new path to node 1. The problem is that even if the AS Path info that node 1 would have was wrong, with the traditional BGP4 during those 30 seconds node 1 would have node 2 as next-hop to D. And node 2 has the correct path. But with Ghost-flushing, the withdrawal triggered by the second path change and the first path change for 30 seconds node 1 cannot reach D.

5.1.5 "Multi" Collection

This collection of models have been constructed from tables dumps of real BGP4 routers on the internet. They come from the Internet Provider topologies and they were found at [26]. They are multi-AS router-level topologies approximating the customer-provider or peering relationships of Autonomous Systems extracted from the BGP routing tables. They were built following this process[26]:

1. Generate a topology from a BGP table dump (such as RouteViews);
2. Prune a given percent of links from the topology, and take the largest remaining connected component;
3. Merge nodes together, choosing nodes with smallest degrees first, until topology is reduced to 1000 nodes;

4. Don't merge if nodes share any peers, and if degree of both nodes is greater than 2.

This models are extremely interesting since they show the effect that the deployment of Ghost-Flushing or Ghost-Buster techniques would have over typical internet provider structures. We had network models of sizes until more than 800 nodes, but they turn out too heavy to be simulated so we stopped at the size of 409 nodes. To create images showing the structure of this models we used the RaceWay Viewer class included in the SSFNet package[1]. In this section we include *fig.E.1* showing the structure of the smallest net. In the appendix images of all the networks can be found.

Size 29, 110, 208 and 409 nodes.

Convergence Times studied Stabilization and E_{up}

Interest Practical.

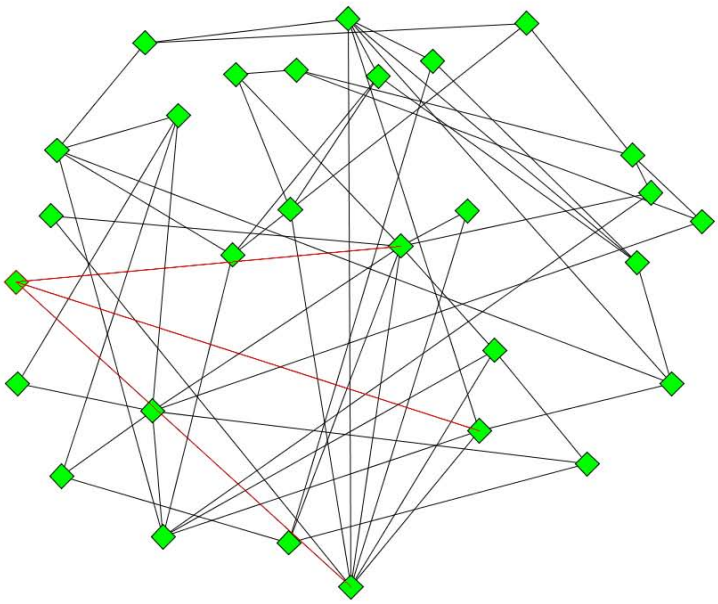

Figure 5.5: "29 nodes structure". Image Generated by RaceWay network visualization tool

5.2 Simulation Results, Split-horizon disabled, No jitter

The results on this section is divided depending on the simulation options (Split-horizon and Jitter) and the topology.

[1] To use it, the way is using the same make file that with the simulations but with the "view" method.

5.2.1 Clique Topology

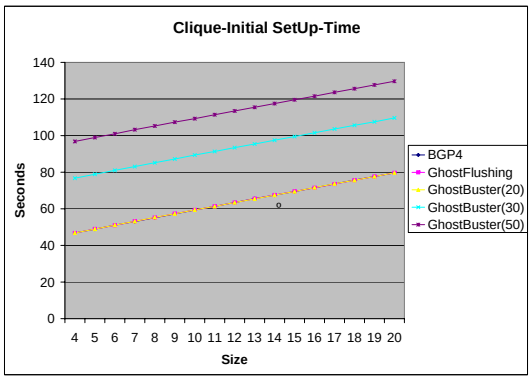

Figure 5.6: Clique Set Up time, no Jitter, No Split-horizon.

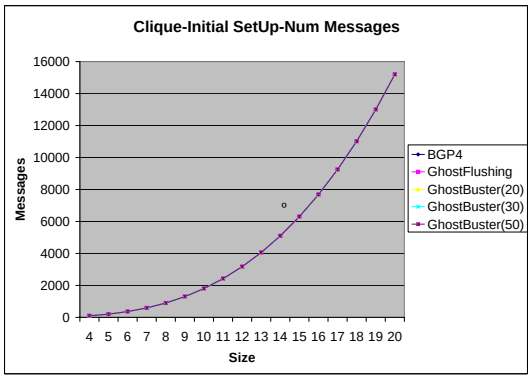

Figure 5.7: Clique Set Up Messages, no Jitter, No Split-horizon.

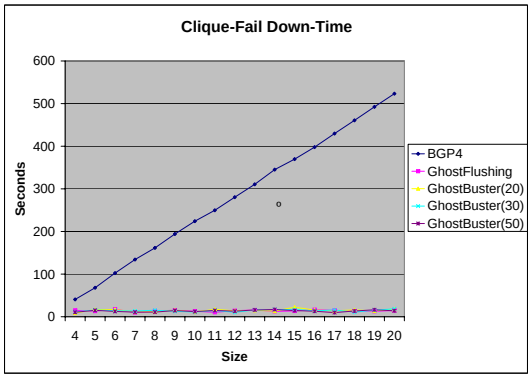

Figure 5.8: Clique Fail Down time, no Jitter, No Split-horizon.

Set Up

Since the ghost-flushing technique doesn't alter the setup situation in a clique (No update to longer paths) the results are the same for messages and time. On the other hand, here we start to notice the drawbacks of the ghost-buster technique, since it delays any update from the time the last update was received, having a negative impact in the E_{up}. The number of messages is the same, since even it updates have to be delayed, the same updates are needed.

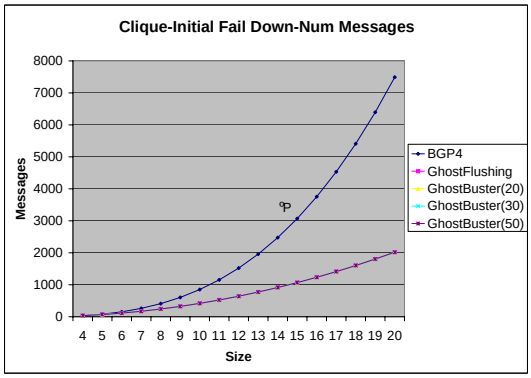

Figure 5.9: Clique Fail Down messages, no Jitter, No Split-horizon.

Fail Down

About the fail down situation. The ghost-flushing starts to show its advantages. While traditional BGP4 seems to have a linear convergence (On the size of the clique) of E_{down}, the ghost-flushing version seems to be constant and keeping a really low value. The ghost-buster technique runs over the ghost-flushing rules, and since the flushing rules eliminate any possibility (By General withdrawal) of a second update blast after the first updates triggering the ghost-flushing rule.

About the number of messages. The traditional shows a exponential behavior. On the other hand the ghost-flushing shows a linear behavior due to the fact that the number of messages sent are just general withdrawals sent by all nodes: $(n-1)(n-1)$ ($(n-1)$ messages sent by $(n-1)$ nodes).

5.2.2 AltPath Topology

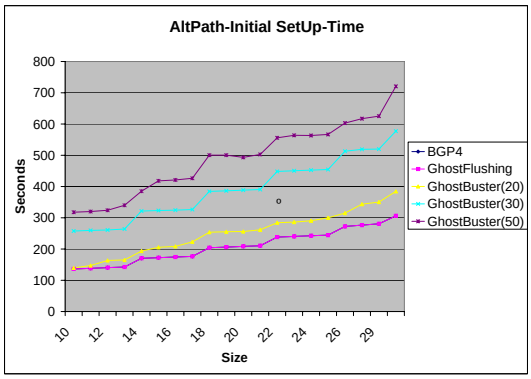

Figure 5.10: AltPath Set Up time, no Jitter, No Split-horizon.

Set Up

Again, like in the previous case, traditional BGP and Ghost-Flushing behave in the same way. On the other hand, the side effect of the delta timer is highly noticeable in this graphics. You can notice performance disadvantage proportional to the value of the delta delay, we can see parallel performances.

The number of messages is the same, even if there are delays. So, no differences in the number of messages.

Fail Over

All the versions show a similar style of convergence, polynomial. But the difference is the degree, while the ghost-flushing shows a lower degree, the traditional BGP shows the highest of all, in a tie with the

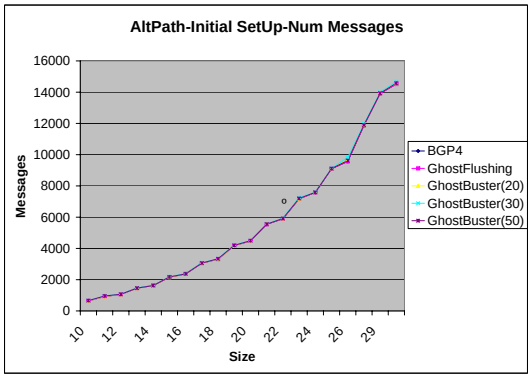

Figure 5.11: AltPath Set Up Messages, no Jitter, No Split-horizon.

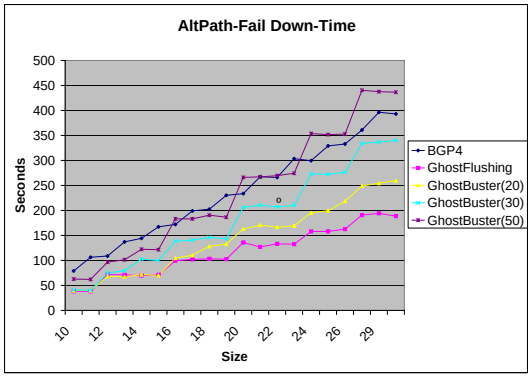

Figure 5.12: AltPath Fail Over time, no Jitter, No Split-horizon.

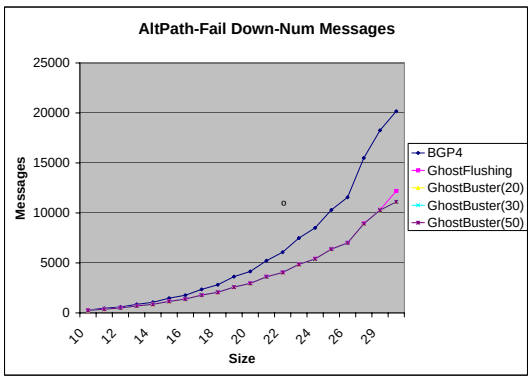

Figure 5.13: AltPath Fail Over messages, no Jitter, No Split-horizon.

ghost-buster with the delta timer value of 50, that grows worse from 24 nodes size. In this case the early elimination of ghost information makes the rules to behave better than the traditional BGP. However the delta timer is a big drawback, even without the ghost-information, update messages are needed in this scenario (Fail over scenario) and they are consequence of other updates.

About the messages on the fail over, eliminating the ghost-information obviously eliminates the reproduction of it in more ghost-information, reducing the number of messages. The performance of all the rules is the same (Not considering the glitch).

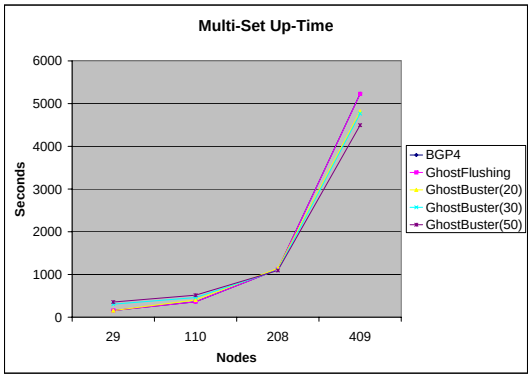

Figure 5.14: Multi Set Up time, no Jitter, No Split-horizon.

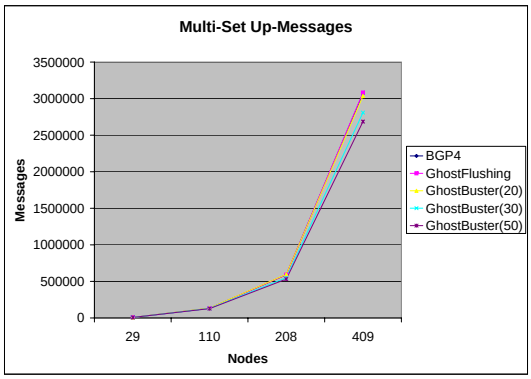

Figure 5.15: Multi Set Up Messages, no Jitter, No Split-horizon.

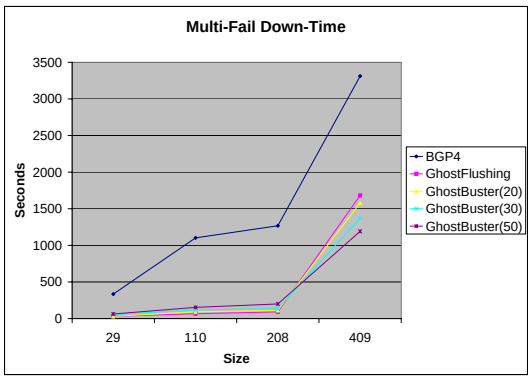

Figure 5.16: Multi Fail time, no Jitter, No Split-horizon.

5.2.3 Multi Topology

Set Up

Looking at the results in this charts we first notice that the values cross when we pass to extremely big networks (From the 200 nodes value). If we look to the smaller values (29,110) Ghost-flushing and traditional BGP behave better than the ghost-buster. Less messages and less time. But when we switch to the bigger values, it x-crosses and the Ghost-buster technique starts to behave better, and, the bigger the delta, the better. It looks like conservative approach in extremely big networks works. The time and messages used spreading paths that won't be used in the end is less when we use a much more conservative approach of ghost-buster.

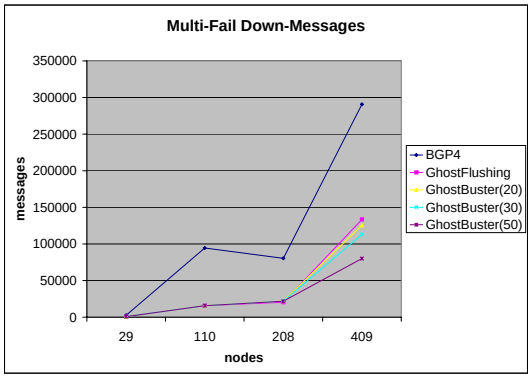

Figure 5.17: Multi Fail messages, no Jitter, No Split-horizon.

Fail Down

Again, we see the same crossing effect. In the smaller node sizes, ghost-flushing is the best version, while ghost-buster was worse than ghost-flushing and BGP4 the worst by far. But in the biggest value (408) Ghost-buster behaves much better (minutes better). Again, it looks that in extremely big networks, the conservative approaches wins again. Parallel results are seen in the number of messages.

5.2.4 White Topology

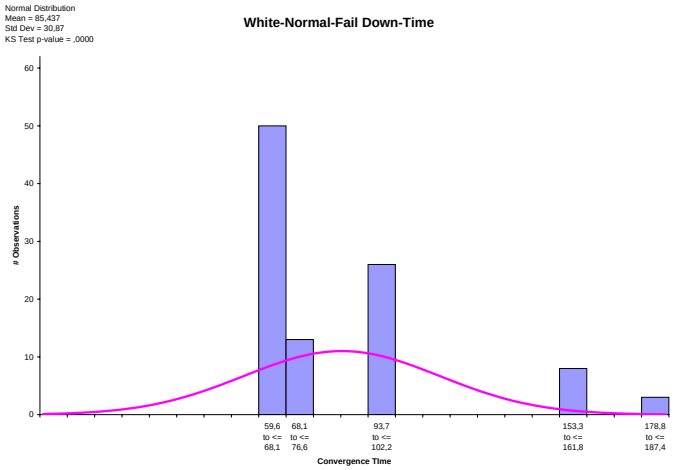

Figure 5.18: White Fail down time, no Jitter, No Split-horizon.

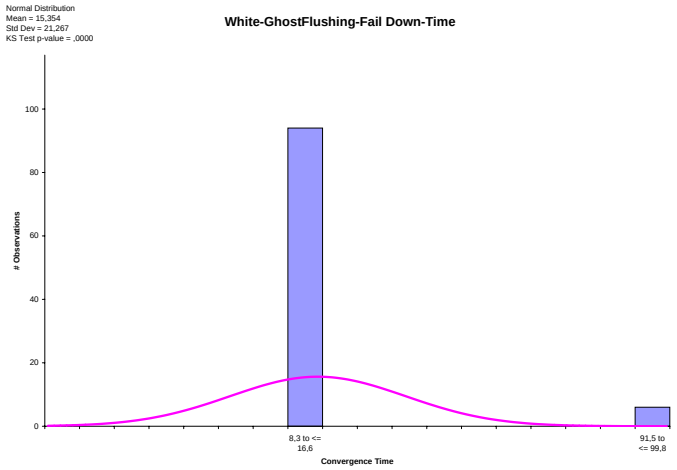

Figure 5.19: White Fail down time, no Jitter, No Split-horizon.

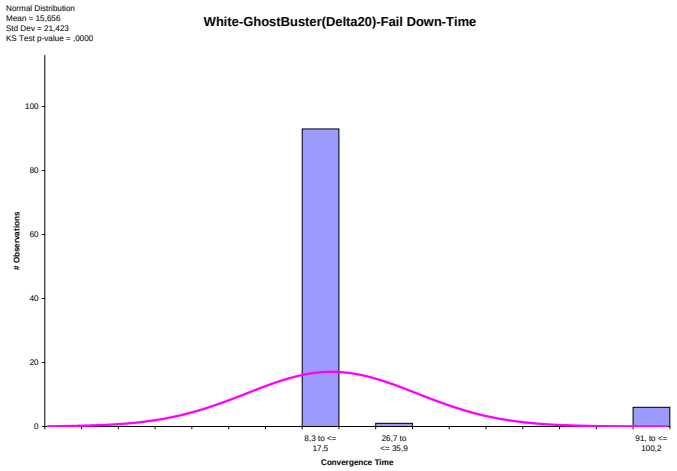

Figure 5.20: White Fail down time, no Jitter, No Split-horizon.

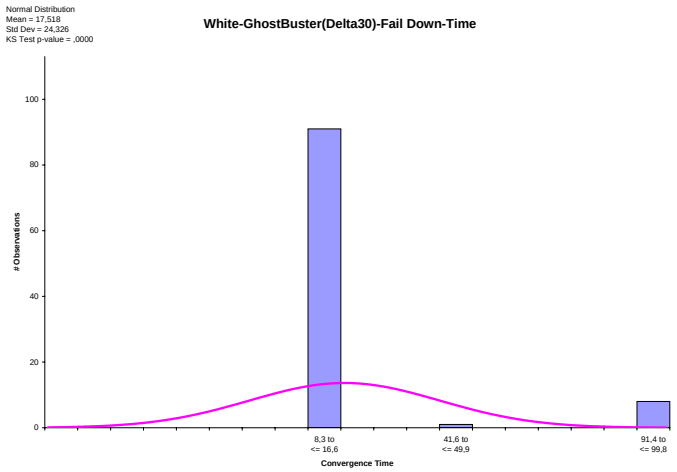

Figure 5.21: White Fail down time, no Jitter, No Split-horizon.

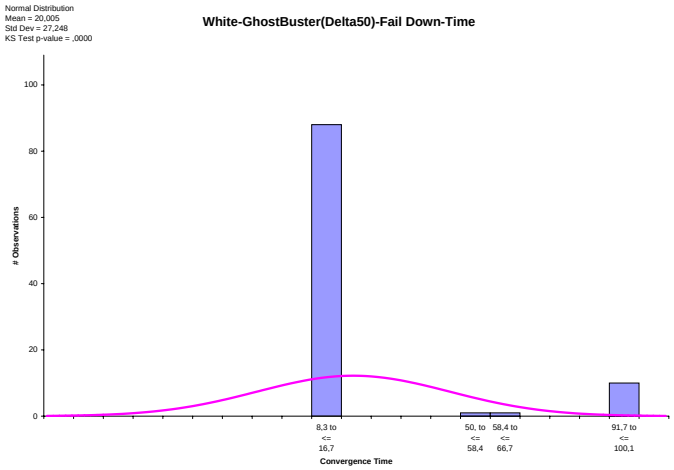

Figure 5.22: White Fail time down, no Jitter, No Split-horizon.

Fail Down time

Since showing the histogram result of all the values studied would be too long we will just show the values about the E_{down}.

As we noticed, the rules behave much better than the original BGP4. The values that appear the most are much lower.

5.2.5 Bad Case

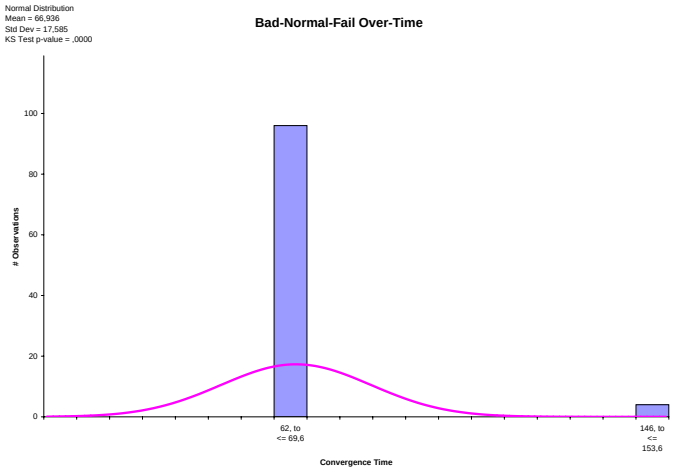

Figure 5.23: Bad Fail over time, no Jitter, No Split-horizon.

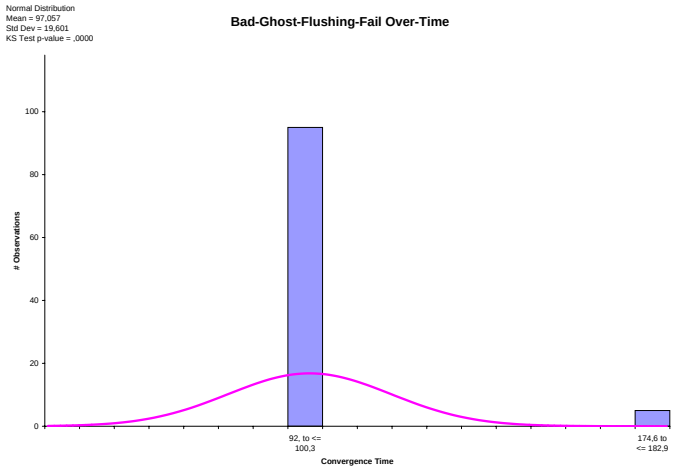

Figure 5.24: Bad Fail over time, no Jitter, No Split-horizon.

Fail over Time

Again we limit the number of histograms to this two cases, traditional and ghost-flushing bgp. In this case we didn't consider the total stabilization of the system, but when the node 1 can access the destination. The result is as show, the MRAI timer of difference, being the traditional BGP a winner in this category,.

5.3 Simulation Results, Split-horizon disabled, Jitter Activated

5.3.1 Clique Topology

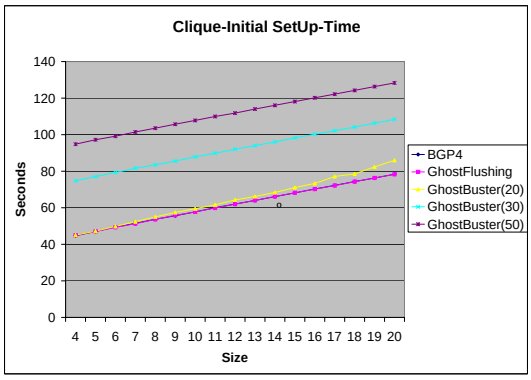

Figure 5.25: Clique Set Up time, Jitter, No Split-horizon.

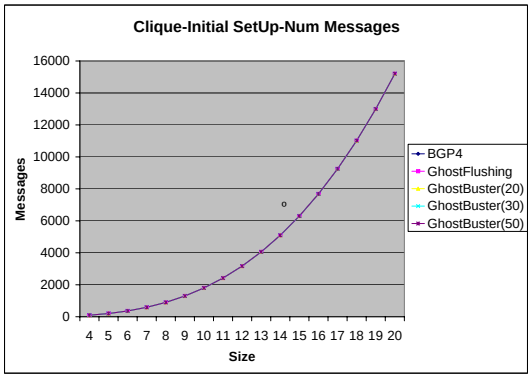

Figure 5.26: Clique Set Up Messages, no Jitter, No Split-horizon.

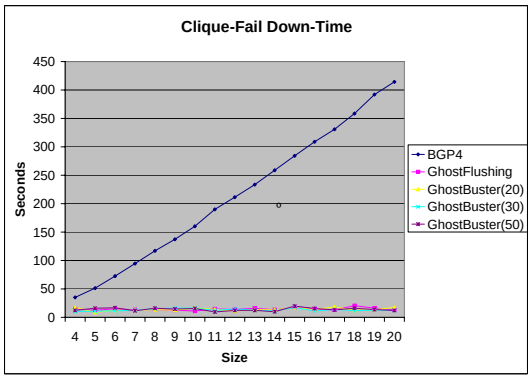

Figure 5.27: Clique Fail Down time, Jitter, No Split-horizon.

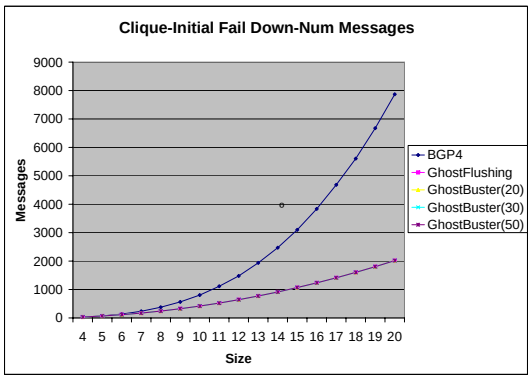

Figure 5.28: Clique Fail Down messages, Jitter, No Split-horizon.

Set Up

Since we get the same results as without using the jitter we won't make any more comment.

Fail Down

The only big difference in the case of activating the jitter is that initially the distance between the rules and the traditional is smaller. However the convergence tendency is the same. For E_{down} is polynomial for the traditional BGP and constant for the rules.

5.3.2 AltPath Topology

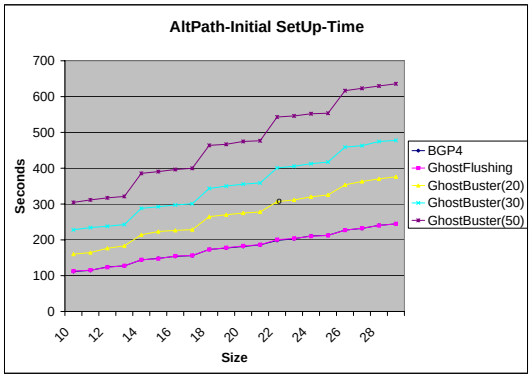

Figure 5.29: AltPath Set Up time, Jitter, No Split-horizon.

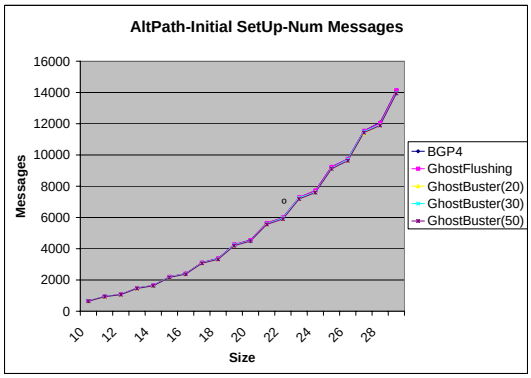

Figure 5.30: AltPath Set Up Messages, no Jitter, No Split-horizon.

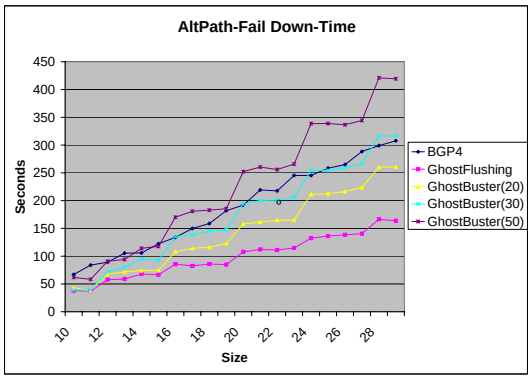

Figure 5.31: AltPath Fail Over time, Jitter, No Split-horizon.

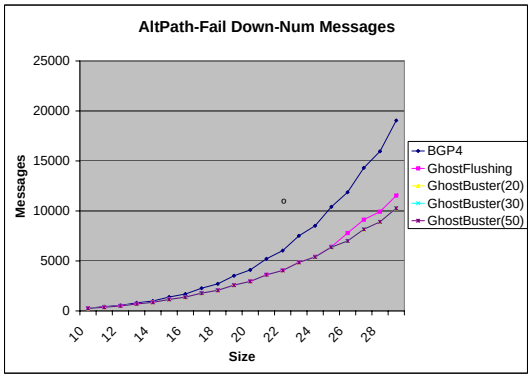

Figure 5.32: AltPath Fail Over messages, Jitter, No Split-horizon.

Fail Over

Again, using the jitter seems to be an improvement of the traditional BGP4 compared with the rule applied version. In further chapters a deeper comparison between using the protocol with or without the jitter will be done.

5.3.3 Multi Topology

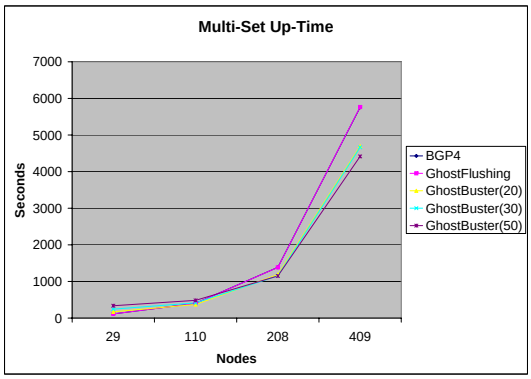

Figure 5.33: Multi Set Up time, Jitter, No Split-horizon.

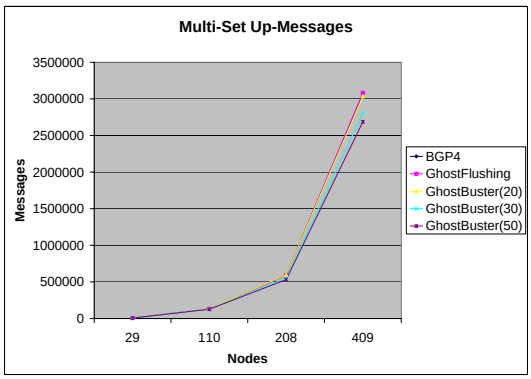

Figure 5.34: Multi Set Up Messages, Jitter, No Split-horizon.

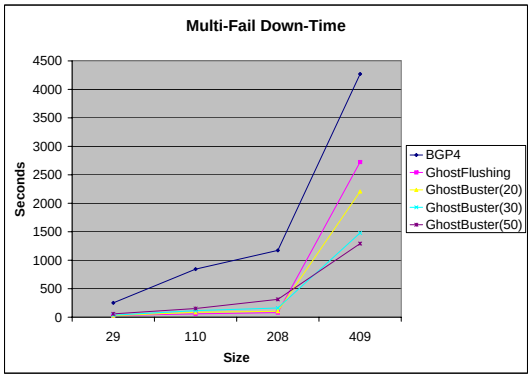

Figure 5.35: Multi Fail time, Jitter, No Split-horizon.

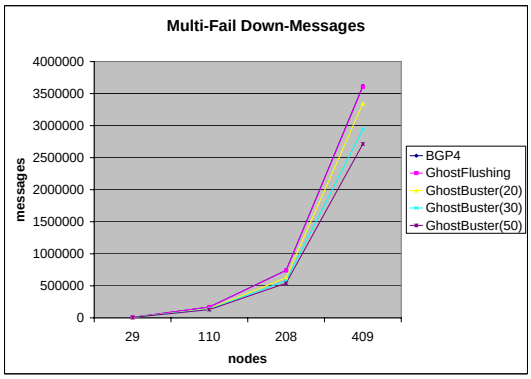

Figure 5.36: Multi Fail messages, Jitter, No Split-horizon.

Fail Down

Again we see and improvement of the traditional BGP4 and the GhostBuster Rules. The ghost-flushing doesn't appear to change after using jitter.

5.3.4 White Topology

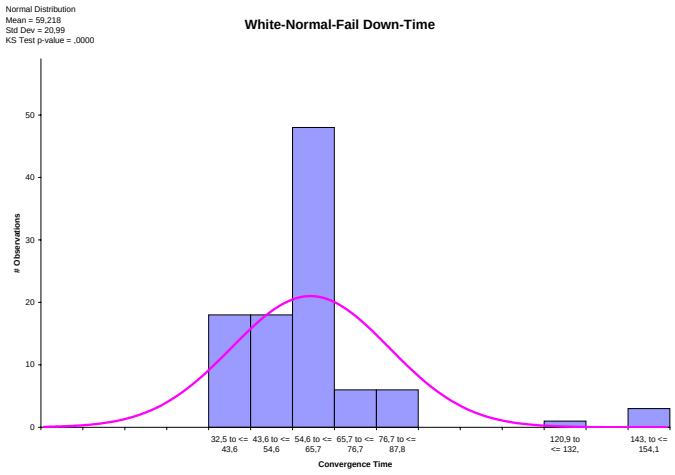

Figure 5.37: White Fail down time, Jitter, No Split-horizon.

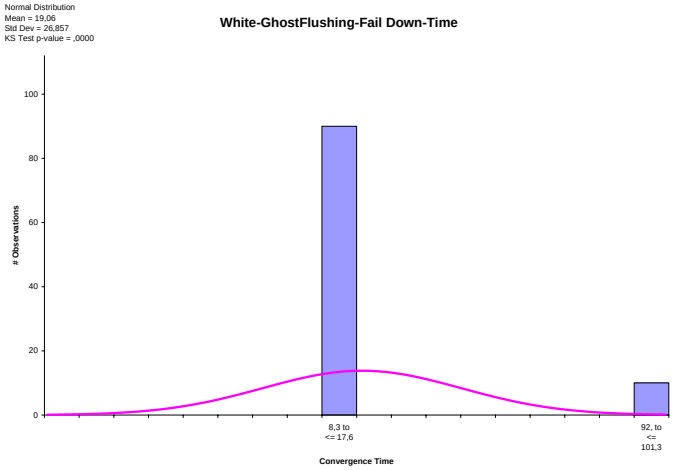

Figure 5.38: White Fail down time, Jitter, No Split-horizon.

Fail Down time

Looking to this Histograms about E_{down} we can see a more stable response with the Ghost-Flushing rule, convergence times vary less. Also they gather around a much smaller time.

5.3.5 Bad Case

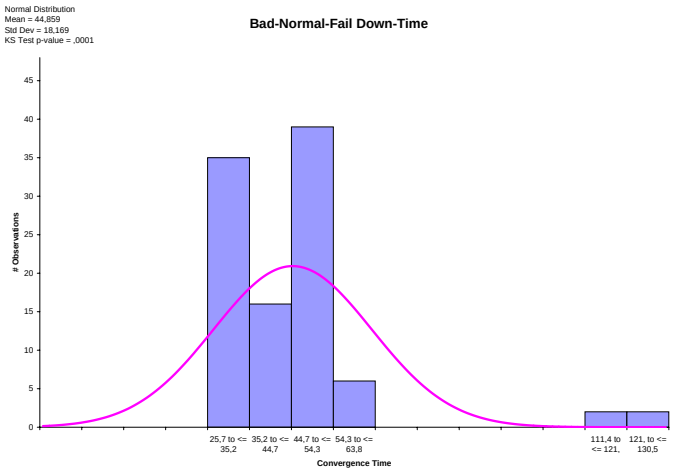

Figure 5.39: Bad Fail over time, Jitter, No Split-horizon.

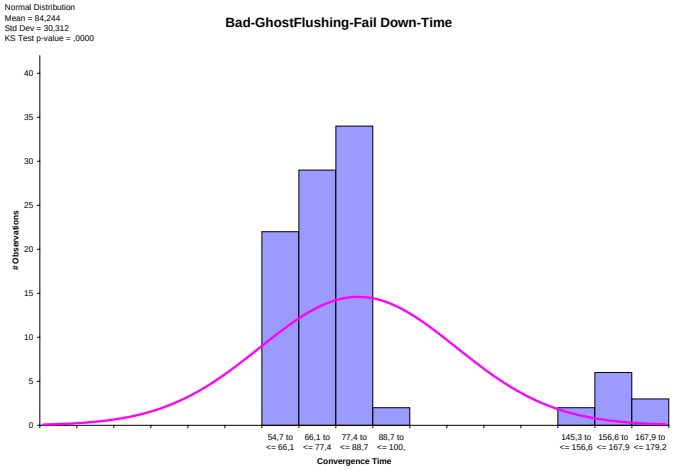

Figure 5.40: Bad Fail over time, Jitter, No Split-horizon.

Fail over Time

From a stability point of view the results are similar, but still, the delay for the ghost-flushing due to the unnecessary flushing of the path is noticeable. This path has a incorrect AS-Path info but a correct Next-hop attribute, what makes it valid.

5.3.6 Comparing between jitter enabled and disabled

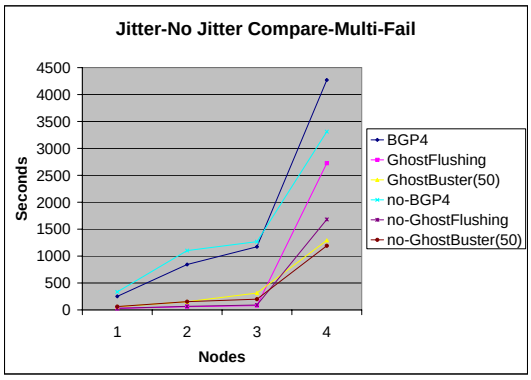

Figure 5.41: Comparing the advantage of using jitter on timers.

We can notice a dramatic improvement en each version of the protocol. The Convergence time in this multi scenario improves in a 10%. So it seems that there are not negative of side effects in using jittered timers on ghost-flushing or other of the rules.

5.4 Simulation Results, Split-horizon Enabled, No Jitter

5.4.1 Clique Topology

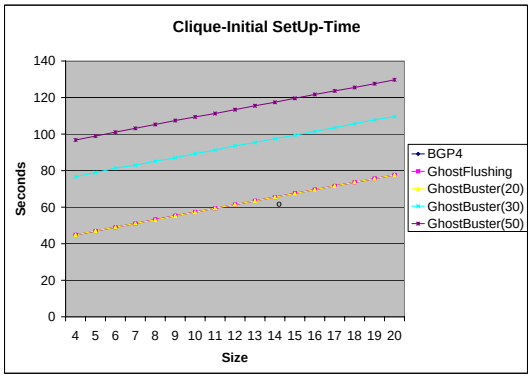

Figure 5.42: Clique Set Up time, No Jitter,Split-horizon.

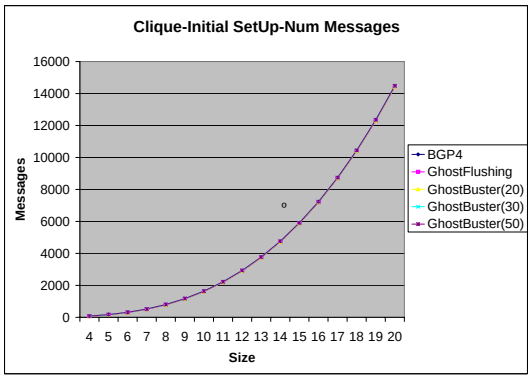

Figure 5.43: Clique Set Up Messages, No Jitter,Split-horizon.

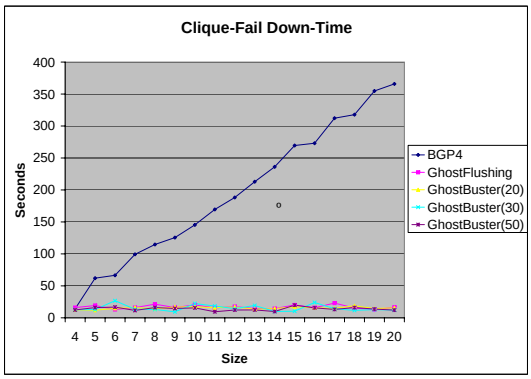

Figure 5.44: Clique Fail Down time, No Jitter,Split-horizon.

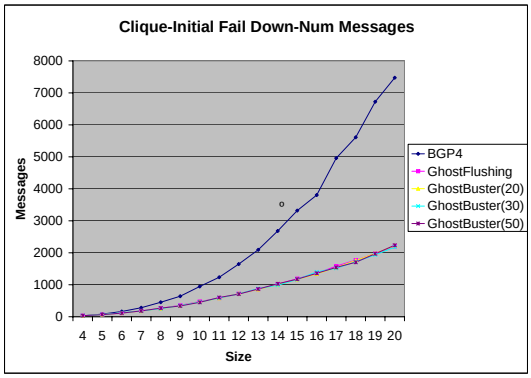

Figure 5.45: Clique Fail Down messages, No Jitter,Split-horizon.

Set Up

We get exactly the same results as in the case that we didn't activate the Split-Horizon.

Fail Down

About E_{down} Something interesting happens. In a 4 nodes clique, activating Split-horizon and using ghost-flushing is the same. Let's explain this. Imagine we have 4 nodes, node number 4 goes down, we have 3 left. N_2 (Node 2) and N_3 chooses N_1 as next hop. N_1 chooses N_2 as next hop. That means a withdrawal from N_2 and N_3 to N_1, and from N_1 to N_2. That triggers that:

- N_1 Has no more routes.
- N_2 Has only N_3 as next-hop, that means a withdrawal to N_3.
- N_3 Has only N_2 as next-hop, that means a withdrawal to N_2.

So there is a general withdrawal because of the split-horizon. However, from the 5 nodes clique, things come to normal. Ghost information reappears and the situation that we found in *fig. 5.8*, a linear convergence for the traditional BGP and constant for ghost-flushing and other rules.

About the number of messages, a parallel case to the convergence time. In the 4 nodes clique, same number of messages as the rules, but then the ghost information appears, reproducing the exponential convergence for the traditional BGP.

5.4.2 AltPath Topology

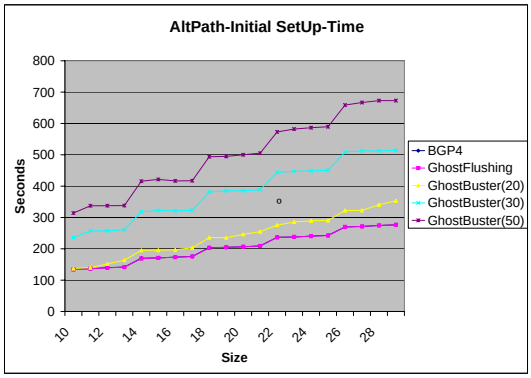

Figure 5.46: AltPath Set Up time, No Jitter, Split-horizon.

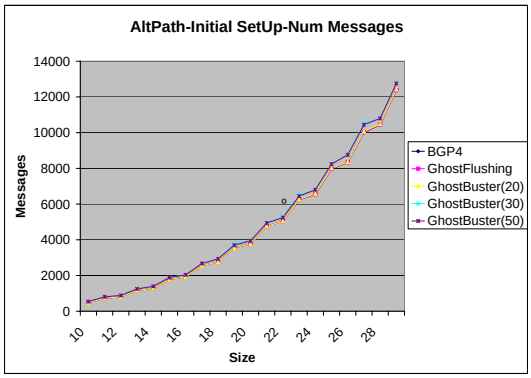

Figure 5.47: AltPath Set Up Messages, No Jitter, Split-horizon.

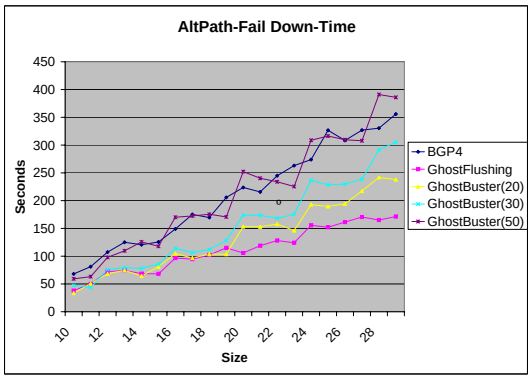

Figure 5.48: AltPath Fail Over time, No Jitter, Split-horizon.

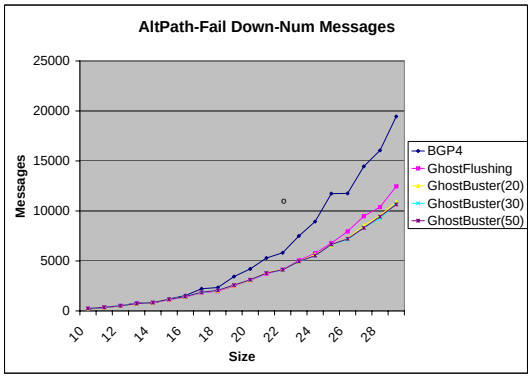

Figure 5.49: AltPath Fail Over messages, No Jitter, Split-horizon.

Fail Over

We notice small differences from the case where the split-horizon was activated. About the E_{down}, until the size of 20 nodes, the ghost-buster rule with a delta of 20 performs the same and sometimes better than the ghost-flushing. But from that point on, the ghost-flushing is the winner.

About the number of messages, the more conservative policy of the ghost-buster rule is noticed in extremely big cliques, but this small differences are anecdotic.

5.4.3 Multi Topology

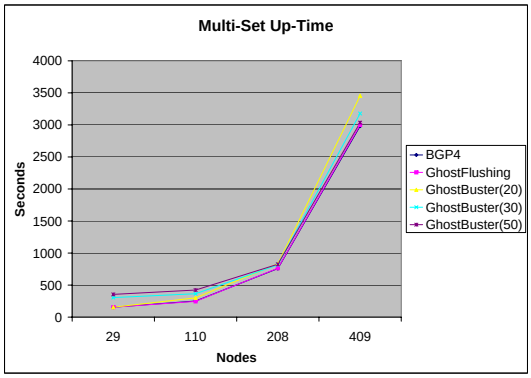

Figure 5.50: Multi Set Up time, no Jitter, Split-horizon.

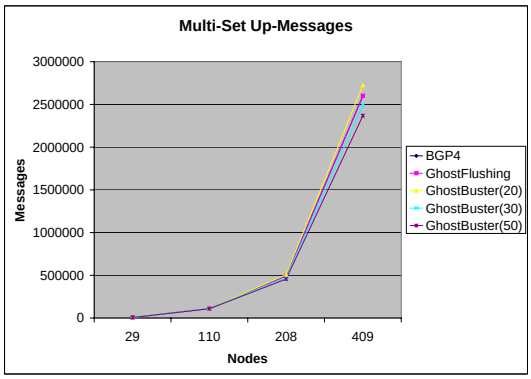

Figure 5.51: Multi Set Up Messages, no Jitter, Split-horizon.

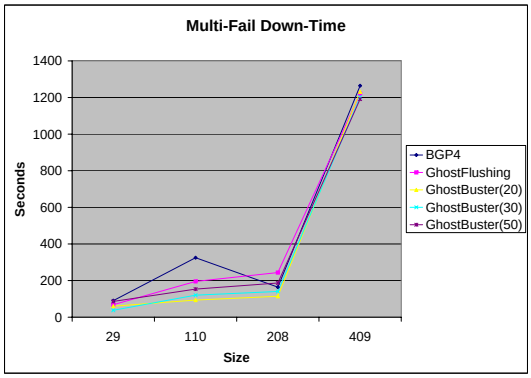

Figure 5.52: Multi Fail time, no Jitter, Split-horizon.

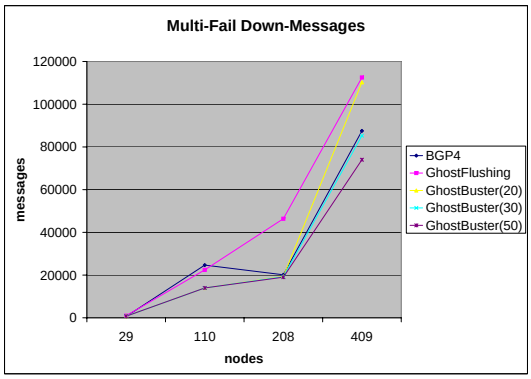

Figure 5.53: Multi Fail messages, no Jitter, Split-horizon.

Fail Down

Looking at the results on the graph, we want to highlight two noticeable facts. About E_{down}, we see a general better behavior of the rules against the traditional BGP. Also we can remark that the combination of the split-horizon with the ghost-buster rules behaves better in the intermediate cases than the ghost flushing. Still the difference is not that big.

Also, the number of messages has that tendency, but even bigger. It looks like the split-horizon boosts the fail-over case. Increasing the number of messages in the ghost-flushing rule. It looks like the conservative approach of the ghost-buster reduces the number of messages.

5.4.4 Comparing between having or not Split-Horizon

In the first place, we notice a general improvement in all the rules. Specially in the traditional BGP4. Still, the other rules behave much better using the split-horizon. Gaining even a nice 30% in the biggest topology.

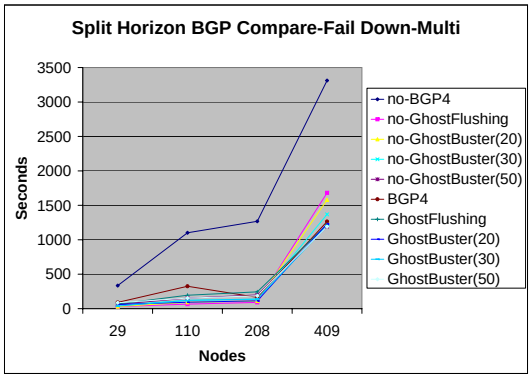

Figure 5.54: Comparing between having or not Split-Horizon.

5.5 Simulation Results, Split-horizon Enabled, Jitter Activated

5.5.1 Clique Topology

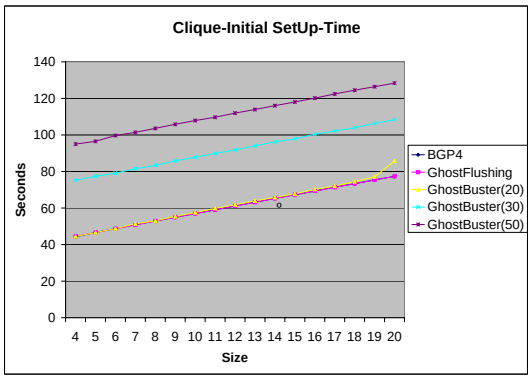

Figure 5.55: Clique Set Up time, Jitter, Split-horizon.

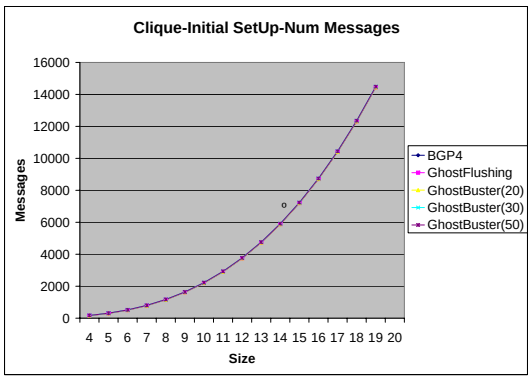

Figure 5.56: Clique Set Up Messages, no Jitter, Split-horizon.

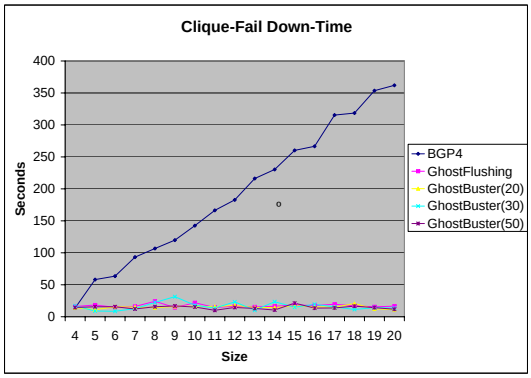

Figure 5.57: Clique Fail Down time, Jitter, Split-horizon.

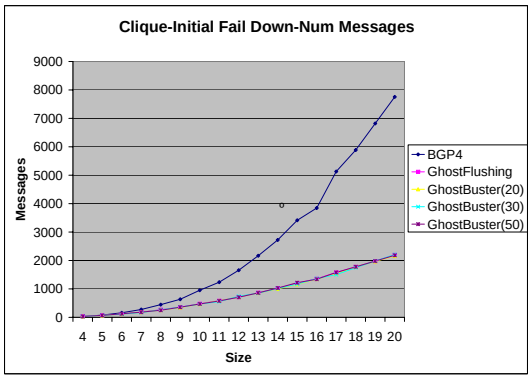

Figure 5.58: Clique Fail Down messages, Jitter, Split-horizon.

In both scenarios we get similar results to the ones found before.

5.5.2 AltPath Topology

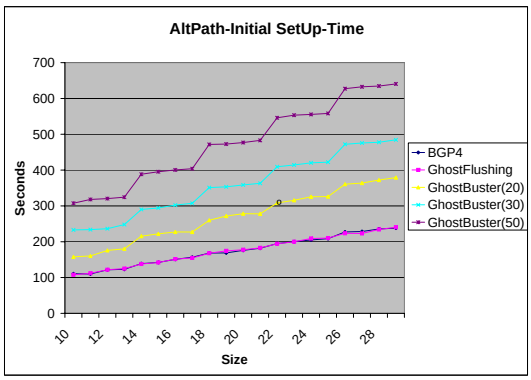

Figure 5.59: AltPath Set Up time, Jitter, Split-horizon.

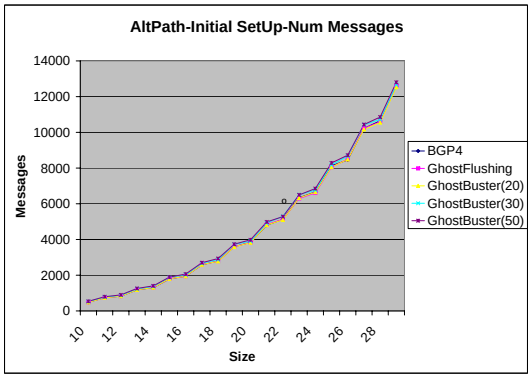

Figure 5.60: AltPath Set Up Messages, no Jitter, Split-horizon.

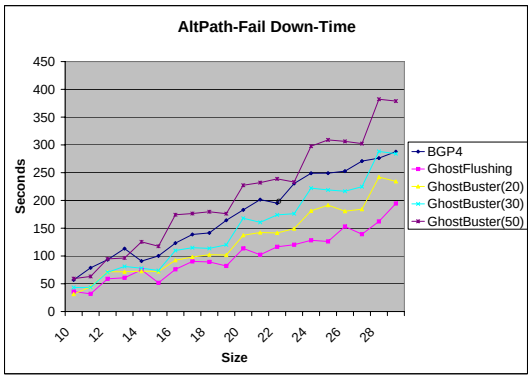

Figure 5.61: AltPath Fail Over time, Jitter, Split-horizon.

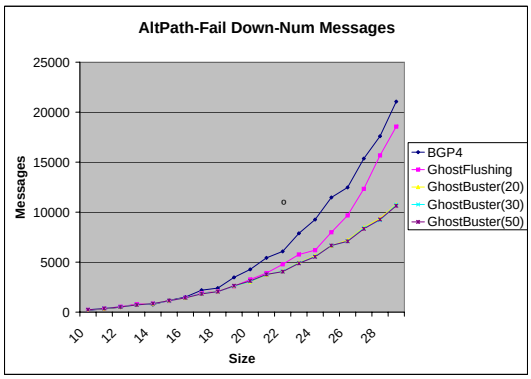

Figure 5.62: AltPath Fail Over messages, Jitter, Split-horizon.

Fail Over

In this case, we see that applying both rules to the ghost-Buster rule has a negative effect on its E_{down}. Ghost-flushing keeps been the best option in this scenario. Later on we will do a general comparison between all the versions of the protocols

5.5.3 Multi Topology

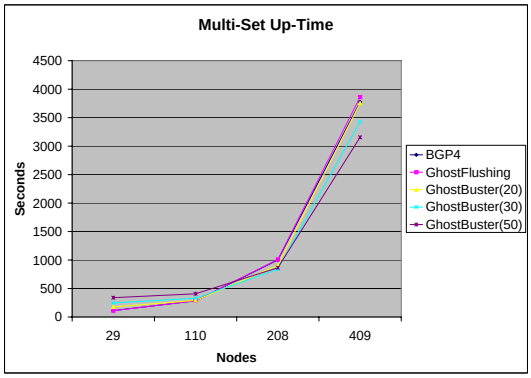

Figure 5.63: Multi Set Up time, Jitter, Split-horizon.

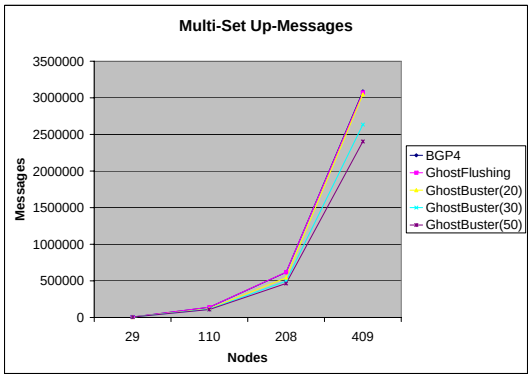

Figure 5.64: Multi Set Up Messages, Jitter, Split-horizon.

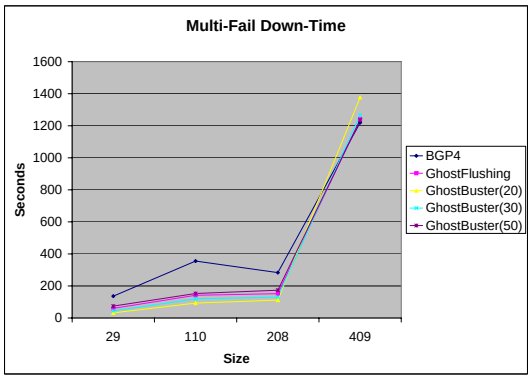

Figure 5.65: Multi Fail time, Jitter, Split-horizon.

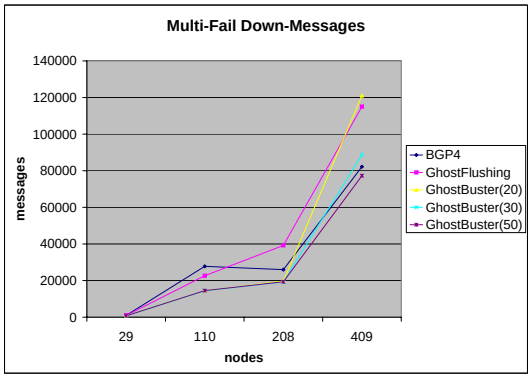

Figure 5.66: Multi Fail messages, Jitter, Split-horizon.

Convergence

We see similar results to the previously obtained. The same tendency crossing appears again. Still if we compare with previous measurements we can find a general improvement in all the BGP versions if we use the split-horizon and the jitter on the timers.

5.6 Comparing all the techniques

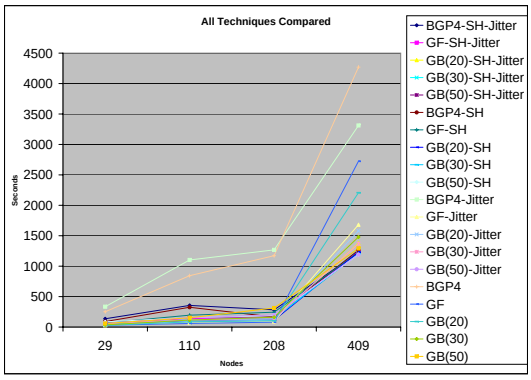

Figure 5.67: Multi E_{down}.

In previous chapters we went through comparing the different techniques with the traditional BGP in many different situations. We repeated the experiments changing the settings of some features inside the BGP4: **jitter applied to timers and split-horizon**.

So in this section we will compare all the techniques with this settings activated and deactivated. The idea is to see which is the best combination of settings. In *fig.5.67* we can see the result of this comparison. For every version, the best performance is found when both split-horizon and jitter are used. We have not seen any "worse" behavior of any of the versions if combined with this settings. It is safe then to use them all together.

5.7 Combining nodes with and without ghost-flushing

One interesting side of this research was to compare what would happen in we combine BGP nodes that use ghost-flushing with nodes that doesn't. Since routers and networks belong to different vendors and institutions, it's hard that all start using ghost-flushing at the same time. This experiment consists to take the scenario where ghost-flushing is more effective, the clique, and start to deactivate the ghost-flushing rule in the nodes.

Cliques between 4 and 15 nodes were taken. The experiment is the same as before. Now, it will be begin with all the nodes applying ghost flushing and then repeated reducing by one each time the number of nodes using the ghost-flushing rule.

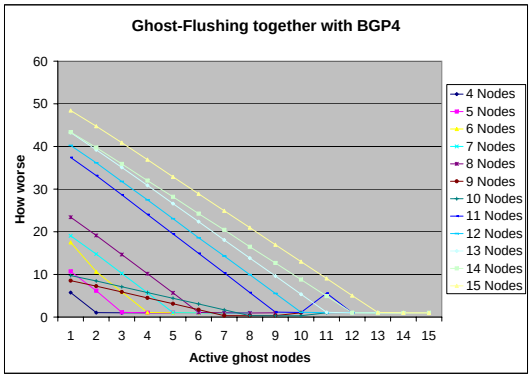

Figure 5.68: Mixing BGP4 with ghost-flushing. More is worse.

The *fig. 5.68* shows the results of out experiment. The content of the figure is confusing so let's explain it. Each line represents the evolution of each clique, beginning (left) with no ghost-flushing nodes and then increasing the number of nodes. As it's natural, each series finishes when the number of active nodes reach the total number of nodes in the clique. The value on "How worse" means how many times bigger is E_{down} with that number of ghost-flushers than the E_{down} got on that clique with all the nodes applying the rule.

As we can see in the figure there is a linear reduction of the E_{down} as we increase the number of flushers. Reaching the point where only 20% of the nodes doesn't use the rule the convergence time is really close to the full ghost-flushed clique. This is really good news if we think of the deployment of BGP4 ghost-flushing on the internet. We won't need all the nodes to use it to start noticing improvement in the performance.

5.8 Conclusions

After all this graphics it is time to summarize a general idea after all the simulations.

5.8.1 Ghost-Flushing

As expected, the behavior of ghost-flushing has been quite good. Compared with the traditional BGP4, it has improved in all aspects (Convergence time and number of messages), no matter the scenario. Only in some situations, applying the split-horizon and the jitter, the traditional BGP4 approached the ghost-flushing performance. However, this only happened in cases of reduced sizes, when the number of nodes was increased the ghost-flushing overtook again the traditional BGP4.

In the **Set up situation**, there is no difference from the traditional BGP. In the **fail over** situation and the **fail down** there is a great difference, the convergence tendency changes from exponential (in the case of the traditional) to linear or constant, depending on the connectivity. It feels like the more connected are the nodes, the more effective is the ghost-flushing rule (Or the worse traditional BGP4 works).

Concerning the effect of this rule depending on the topology, it is clearly noticed that the more "connected" the nodes are, the more effective it is. We can find highly connected situations in scenarios related to ISP interaction, where many ASs connect between them as shown in [18].

Also the **tests of mixing ghost-flushing** nodes and traditional BGP4 shows a great result. Even if not all the nodes uses the rule, the performance of the systems improves. There is a linear relationship between number of nodes and the improvement of the performance. This is a great result for the future deployment of the rule.

About the implementation. It turns out to be extremely easy and it doesn't look to need extra cpu time (noticeable) or memory usage.

Looking to all the results and conclusions, we can say that **it's worth it to implement this rule** later on GNU Zebra. The results are quite good and the drawbacks look small enough to do it.

5.8.2 Ghost-Buster

The results about the ghost-buster rule are quite surprising. Initially, when we were running the first simulations, we found the expected results, a performance between the traditional BGP and the ghost-flushing rule. However, when we started to simulate big models and to apply the split-horizon rule or the jitter on the timers, **the performance of the ghost-buster boosted.**

Let's begin with the **Fail down**. There are cases where the ghost-buster with the biggest delta timer (50 seconds) outperforms the rest of the configurations. The model where this effect appeared is the multi topologies. When we test over the biggest models, the ghost-buster rule performs better than the ghost-flushing in both time and messages. It looks like the conservative approach in high interconnected situations, when long and numerous paths exist, ghost-buster performs better.

Similar situation appears in the **set up scenario**. Again, in extremely big scenarios, the ghost-buster rule performs quite well in number of messages and time.

Looking to this results we can try to extract some **general conclusions**. It looks like that, depending on the point of view, the ghost-buster performs better or worse. If we take in account the time that takes to stabilize the whole network, the ghost-buster performs quite well. However, if we look the small picture, if we look to small parts of this networks (Smaller models) the ghost-buster performs worse.

We didn't run any simulation of how the big models (400) nodes perform when we reduce the number of ghost-busters in the network. It is just a matter of real time. Simulate once the 400 nodes scenario takes almost an hour, we would need to run 400 simulations more than once to get stable results, and thats a lot of time.

Something important about this rule is that it has **high needs of cpu and memory**. Every new path received needs a timer that allocates memory, and produces and event when expired (Even if no effect needed). This can have a bad impact in the router performance.

Looking to all this conclusions we see good points and bad points. We see a **bad performance in medium, small scenarios** (less than 200 nodes) for the set up and fail situations. This makes us to take the decision of not implementing this rule.

Chapter 6

Applying the Ghost Flushing Rule to Zebra Routing Software

After the analysis of simulations from the different modifications proposed for BGP4, we came with a clear winner, the ghost-flushing. Then, it was a moment to do an implementation over a real routing software. So, an open source free software was needed, we looked at the market and we saw a few options: Zebra, Quagga, GateD and OpenBSD's bgpd.

GateD was discarded when we checked out its web site [3] and found out that it wasn't an open source project anymore and becoming commercial software.

Quagga[8] is a spin off from Zebra routing software, a new community has been born trying to implicate more people in the development of this software. However, we preferred to stick to the original version, since it's been running for a longer time and more documentation was available (Not much though).

We learnt about **OpenBSD's bgpd** after we started working with Zebra. The main problem we had is that we had to set up an OpenBSD environment to make it work. Also the portability was in issue, we were not sure about how this software could be deployed over other operating systems, if possible.

Finally **GNU Zebra**[4] was the strongest remaining option. Strongly modular and with a long running life (Since 1996) it looked like a good option and quite used (From our experience in the development mailing list from Zebra). There were also positive opinions about the bgp module in zebra by the routing community[34, 23].

6.1 Zebra as a Routing Software

GNU Zebra is a free software that implements a set of TCP/IP routing protocols. It includes the most used ones like BGP4,RIP (v1 and v2) or OSPFv2. In the case of BGP4 it follows the RFC-1771[28] implementing most of its features. One of its most attractive characteristics is its modular approach. Every routing protocol is implemented in a different daemon, being the Zebra daemon their link to the kernel routing table. This made the modification of bgpd an issue that didn't imply interfering with the code related to all the rest of routing protocols.

On its background, the company IP Infusion Inc. is supporting this project hosting since it's the personal creation (At the begining at leat) of its CEO, Kunihiro Ishiguro.

6.1.1 History

(From GNU Zebra page[4]) The Zebra project began in 1996. The idea for Zebra originally came from Kunihiro Ishiguro, who had been working at NIS, an ISP joint venture between British Telecom and Marubeni. Working for an ISP, Ishiguro had realized a great need for a new type of quality routing software. It was at this time that Ishiguro met Yoshinari Yoshikawa. Yoshikawa shared Ishiguro's vision for a new routing engine and they decided to combine resources to create the world's first routing engine software based on the GNU General Public License. This entity, called the Zebra Project, consists of the business expertise of IP Infusion combined with the technical skills of the world's top networking engineers and a commitment to offer top-quality free software routing engine.

Today, Zebra is nearing completion and the release of version 1.0. The vision of a free routing software that can respond quickly to changes in technology and offer functionality that users require is now even more critical. As internet use explodes, no longer can one company or proprietary software provide all the answers. In the case of Zebra, the mailing list and comments of users and engineers have evolved the software into the form it has today.

6.1.2 Features

The strongest feature is it's **modularity**, each routing protocol is implemented in different daemons, this includes some of the various advantages of this software package:

- **Easy to modify**. In our case, we wanted to modify just the BGP4 part of this software. The modular approach allowed us to do it without touching any code related to other routing protocols or the interaction between the suite and the operating system. As a result the range of code to be analyzed was extremely reduced.

- **Asymmetric Load support**. An overload or heavy traffic in one of the routing protocols doesn't have the same impact in the rest of protocols as there would be in a monolithic approach.

- **Reliability**. In case of a problem with one of the protocols or its daemon, the fault will be limited to that protocol, not affecting the rest of the daemons. Also, this fault can be analyzed and corrected without taking the whole router off line.

- **Free and Open Source**. Obvious and compulsory feature for this thesis, since our intention is to modify the behavior of routing suite. Also we have to consider that a network situation can require special routing capabilities. This suite is perfect since it can be easily modified and it's free. In some cases, a router build with a "conventional" computer and Zebra can be a inexpensive alternative to the well known embebed solutions[34]. Fully upgradable and customizable.

About what **routing RFCs it supports**. Concerning BGP4, it fully supports the RFC-1771[28]. However it also implements functionalities included in other RFCs like Route Reflection and Confederations[30, 25] and route damping[16].

It is also **IPv6 aware**, it can work in both ways, IPv4 and IPv6. About the **operating system**, it supports FreeBSD, NetBSD, OpenBSD, GNU/Linux over IPv4 and IPv6. There are plan to support also Solaris. Also the list of **routing protocols included** is: BGP-4, RIPv1, RIPv2, OSPFv2 and OSPFv6.

The **configuration system** tries to clone the notation used on the Cisco routers. This is an advantage in case of previous experience with that brand. It can be configured in a terminal way and it includes and shell interpreter (vtySh) that makes possible writing configuration scripts for the Zebra software, since we can write the terminal commands in a text file and send them directly to the zebra daemons.

Also, it shows some **disadvantages** (Obviously, if not, Cisco would be out of business). In the first place, its modular architecture doesn't allow a communication between its routing protocols as fast as in a monolithic approach (The communication functions rely on sockets). This problem also extends to the kernel routing table, it cannot compete with other options like embebed routers.

6.1.3 Setting Up a Zebra/BGP speaker

In this section we will try to create a small guide line of how to build a BGP speaker with a personal computer. First we chose the **operating system**, FreeBSD is commonly accepted as one of the strongest free Unix/Linux option for networking work, so this was our operating system to run Zebra on.

Installing the software means to follow the typical procedure of installing an open source package, compiling and installing using the configure script and makefiles included on the suite. After this is done a proper configuration file has to be written. An example can be found in the appendixes.

The file we can find in *fig. B.1* configures a router which interface to be known as a router has the 192.168.0.1 IP. Its AS is number 1 and it has 5 neighbors, in the ASs 2 to 5. It advertises the network 130.240.0.0/16 to all of its neighbors. This file belongs to our modified version of Zebra, so it has the *bgp ghost-flushing* entry, indicating the application of this rule is active. Also the log messages of this rule(*debug bgp ghost-flushing*) will be registered.

You can place this file in */etc/bgpd.conf* and it will be automatically loaded when the bgp daemon is started (If not specified otherwise). To automatically start bgpd when FreeBSD boots, we can include a script executing bgpd daemon in */usr/local/etc/rc.d*. Something to notice is that, if we want bgpd to use the usual port (169) it has to be executed with root permissions.

6.2 Architecture of Zebra Routing Software

This chapter will try to give a small idea of how Zebra is structured. This contents are mostly based on [14] and our experience modifying the software. Intended to be a guideline for future modifications of GNU Zebra.

6.2.1 Modules, Daemons and Sockets

As we have pointed out before, **Zebra is a extremely modularized software**. Every task is run by a different daemon. Routing tasks are owned by one different daemon per routing protocol. Also, the function of exchanging routing information with kernel routing tables and the protocols is also done by an exclusive daemon. This helps to divide the code depending on the functions it has to do on complete different modules. The communication between daemons runs over sockets and the protocol used is called Zebra Protocol.

Let's make a small example. In *fig. 6.1* we have a zebra working with three daemons: one for BGP, other for OSPF and another for RIP. bgpd learns Route A through its BGP session with other router. After doing the BGP procedures around this route ,it chooses it and wants to use it. So bgpd daemons sends it through a socket to the Zebra daemon. bgpd, all the routing daemons and Zebra talk a common language: Zebra Protocol. So bgpd uses the Zebra Protocol to transmit Route A to the Zebra Daemon. Then, the Zebra daemon will put the route in the kernel routing table and distribute between the other routing daemons, for example ripd, the one for RIP. Again, to do this the Zebra daemons uses the Zebra Protocol. Then, ripd can spread the learnt route between the hosts it can reach using RIP.

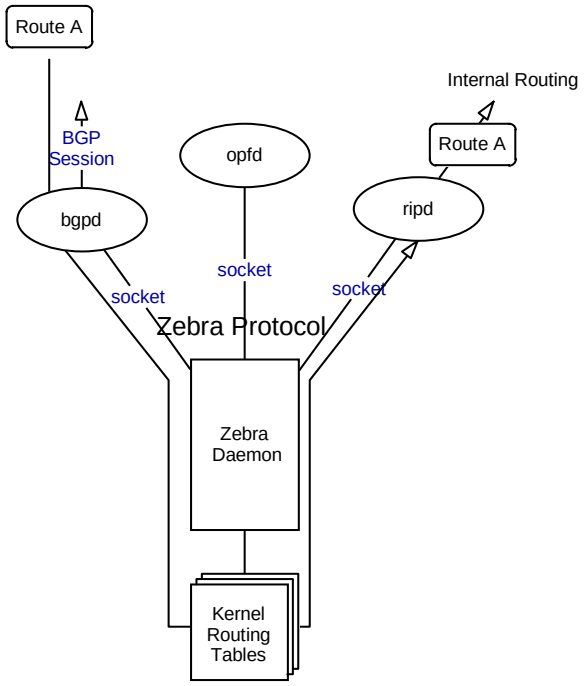

Figure 6.1: Zebra modular structure. All routing information has to pass through the Zebra daemon.

Something really interesting about this approach is that there is no need for all the daemons to be in the same machine. We can have a machine that makes the function of BGP speaker, it would have bgpd

running. But the Zebra daemon can be in a different machine, which routing tables we want to be up to date. Both daemons can connect remotely by sockets. This can slow down the route spread but can divide the load derived from the different functions here shown. For example, we may want a machine to do the gateway function between our network and the internet (So it would run the Zebra keeping its routing table in good state) but leave the BGP transactions to other machine(s). Also we can have machines with different hardware settings, more close to the function they have to do in our network.

6.2.2 Zebra Code Structure

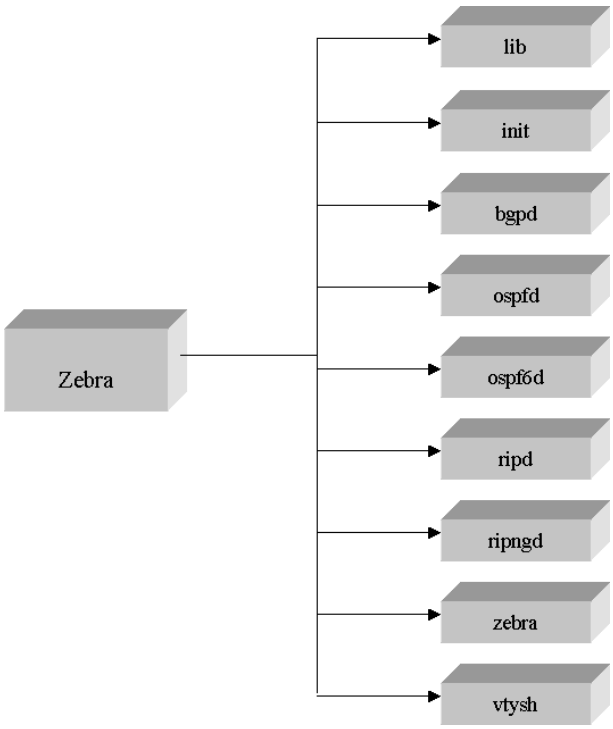

Figure 6.2: Here we find the directory structure for the code of Zebra.

In *fig.6.2* we can see the code tree in Zebra. In bgpd, ospfd, ospf6d, ripd and ripngd we can found all the code related to the daemons managing the respective routing protocol. We will deeply analyze the bgpd later. Something important for this work is that Zebra is completely written in C, that's good and bad. As an application with a big interaction with operating systems, C looks like an adequate language. However, considering there is no a precise documentation of the code (There is about the architecture, but not about the code), C is not the best language to analyze and to find out what it does.

In **lib** all the code related with:

- Connecting to the zebra daemon (zclient.h).
- VTY management (access method (telnet), terminal mangling and cli control).
- Command registration Access lists (commands and functionality) (filter.h).
- Prefix lists (commands and functionality) (plist.h).
- Route maps For becoming a daemon Logging Linked lists, vectors and hashes Memory management, including cli.
- Cooperative multithreading (using select(2)).
- FSF getopt() & regexs.
- MD5.

- Interface structs and functions, including zebra protocol functions (if.h).
- Socket mangling (sockunion.h and sockopt.h).
- IP v4&6 route tables And some internally used functions, like serialization support routines for the zebra protocol.

This all functions are common to all the protocol daemons so here they are coded. If we wanted to implement a new routing protocol, then a big part of the functions needed for such a task are included here.

Zebra directory includes the code about the Zebra daemon. vtySH includes the code for the shell interface that can be used against the console terminal of the Zebra software.

6.3 Architecture of bgpd

In this chapter we will try to show how a daemon works in Zebra. Also, we will study in detail the code related with the BGP functions relevant to the modifications needed to implement Ghost Flushing. This chapter is mostly based on[36, 14].

6.3.1 A Thread's life

All events in Zebra are managed by threads. They are cooperatively-multitasking threads. To create and event, or to set a event in the future, every daemon has a collection of FIFOs where threads can be placed with the function to execute when the event is handled. To do this functions there is a collection

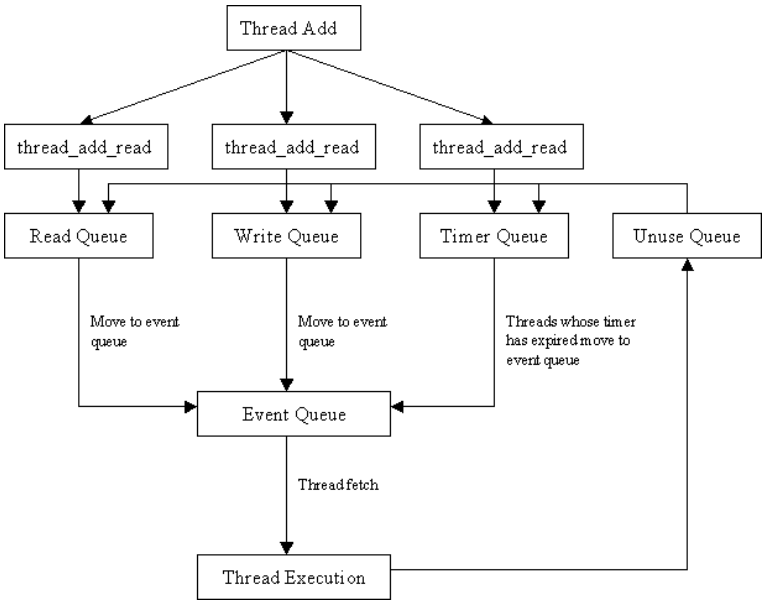

Figure 6.3: Thread life-cycle inside of any of the Zebra Daemons.

of functions:

- thread_make_master (), to create the master thread that will manage everything.
- thread_add_read(), creates a thread that wants to read from a socket to the out world.
- thread_add_write(),creates a thread that wants to write on a socket to the out world.
- thread_add_timer(), creates a thread that wants to be executed in a time set when this function is called.
- thread_add_event(), creates a thread associated to an event.

- thread_cancel_event(), destroys a thread associated to an event.
- thread_fetch(), thread_call(),thread_execute() are used to manage the execution of a thread.

From this list we distinguish four different FIFOs for the threads: one for the ones reading from a socket, another for the ones writing, the one for the handle of different events and finally one for timer based actions. If we want something to happen we just have to create a thread and place it in the corresponding FIFO.

For example, to manage the MRAI timer, we can use the thread_add_timer() to create a thread associated to this timer. We just need to define a function with the actions to be taken when the timer expires. We pass it together with the initialization value of the timer to the thread_add_timer() and done.

6.3.2 Good Daemons speak Zebra Protocol

The file zclient.h defines zclient, the API to talk with zebra. BPG has the file *bgp_zebra.c* that implements the functions corresponding to that API. In that file, the call backs to create/destroy/configure the different interfaces used by the router and route management are implemented.

This file give a series of functions used by the rest of the daemon to easily communicate with the Zebra daemon.

```
void bgp_zebra_announce (struct prefix *, struct bgp_info *, struct bgp *);
   // To send routes to Zebra.
void bgp_zebra_withdraw (struct prefix *, struct bgp_info*);
   // To eliminate routes from Zebra.
```

Inside this part, also the functions that will inject routing information from the Zebra daemon are defined and implemented.

6.3.3 bgpd source files

We now will present all the relevant (for this work) source files, showing their functions.

- bgp_advertise.c bgp_advertise.h: All the code related with sending messages to other BGP4 speakers.
- bgp_aspath.c bgp_aspath.h: Definition of the ASPath types and manipulation functions.
- bgp_aspath.c bgp_aspath.h: Definition of the Attributes types and manipulation functions.
- bgp_filter.c bgp_filter.h: All the filtering policies are implemented here.
- bgp_fsm.c bgp_fsm.h: Here it can find the state machine.
- bgp_main.c: Starting of the bgpd daemon begins here.
- bgp_packet.c bgp_packet.h: Here all the code related with the management of packets going in and out of the BGP socket.
- bgp_route.c bgp_route.h: Route management. Here we can find the **decision process.**
- bgp_table.c bgp_table.h: Tables definition.
- bgp_vty.c bgp_vty.h: Definition of configuration commands/assertion.
- bgp_zebra.c bgp_zebra.h: Zebra communications.
- bgpd.c bgpd.h: bgpd structure definition. The general functions are here.

6.3.4 A bgpd Daemon is born

In this section we will show what bgpd goes through since it is starts. In *fig.6.4* we can see all happening (Most relevant) when bgpd daemon starts. First the bgp_master, a structure that manages al the possible BGP instances, is created. It also includes the main excution thread.

After, the command line parameters passed when the daemon is invoked are processed. Then, memory allocation for the future initializations. Following, the creation of the bgpd structure (bgp_init), it includes the invocation of the configuration VTY environment. After that, the main() function continues. The most important points (For this work) are two: First the configuration file is read an passed through the VTY interface. It means that the configuration file are just VTY commands feeded to the terminal system here. This also means that modifying the configuration system for the file and the terminal are the same thing. The second important point is the end of the main function. It's the starting of the FSM and the eternal execution of the main thread:

```
 /* Start finite state machine, here we go! */
  while (thread_fetch (master, &thread)) 281     thread_call
(&thread);
// This piece of code is executing all the time the threads
// associated with all the events inside bgpd.
```

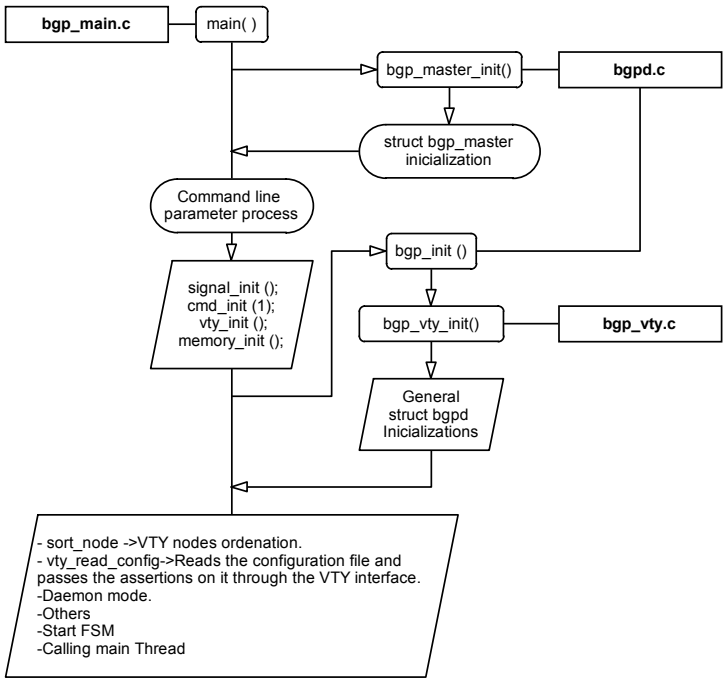

Figure 6.4: bgpd daemon initialization process.

6.3.5 From an Update Arrival to an Update Departure

This section will show a small schema of the update handling/emission system. A part of the BGP implementation critical for future modifications done in this work. In *fig. 6.5* we find the first stage of it, how **bgpd handles a new update message** finishing in the moment when the decision process is run.

The decision process (*fig. 6.6* is really important since that's the place where we will have to do the modifications. Also it shows how to send a withdrawal, something we have to learn to do to implement the ghost-flushing rule. We see that withdrawal sending is done by the function: **bgp_adj_out_unset()**.

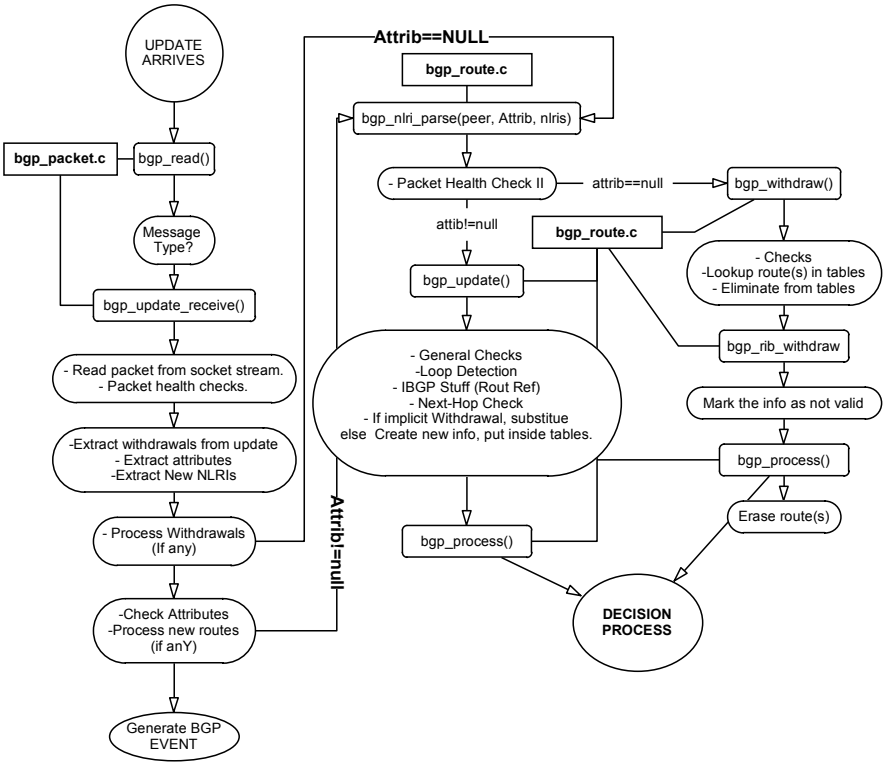

Figure 6.5: Zebra's bgpd,handling an incoming update message.

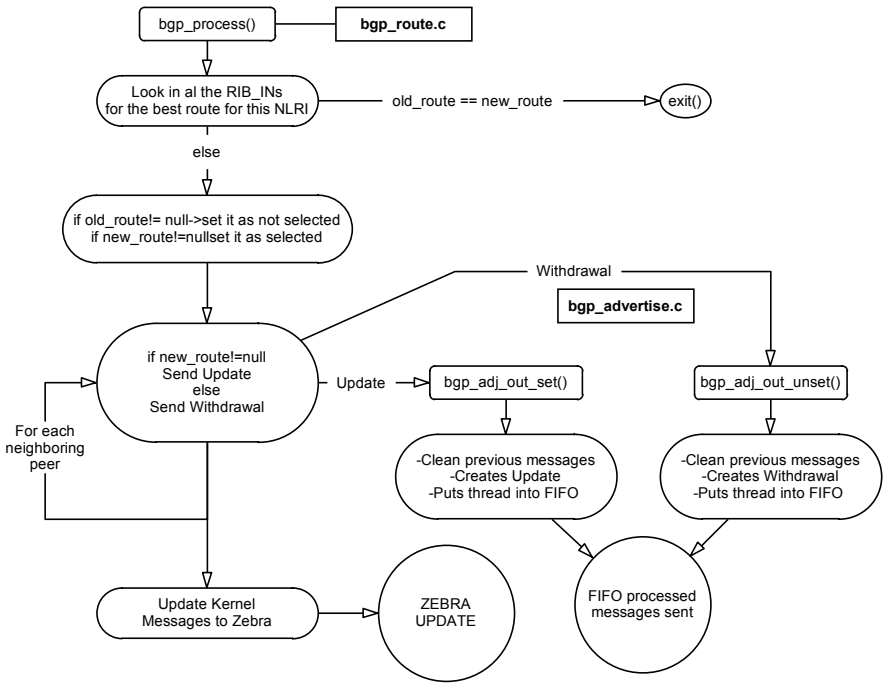

Figure 6.6: Zebra's bgpd decision process (Simplified).

6.3.6 The VTY interface

New functionalities are going to be implemented, so also, a way to activate and deactivate them has to be included. The daemons configuration is done through the VTY interface. Here we'll try to illustrate how it works.

In the first place, **commands in the VTY are organized in a tree structure**. There are two kinds of nodes, environments (The ones connected to and from other nodes) and commands (Leaves of the tree).

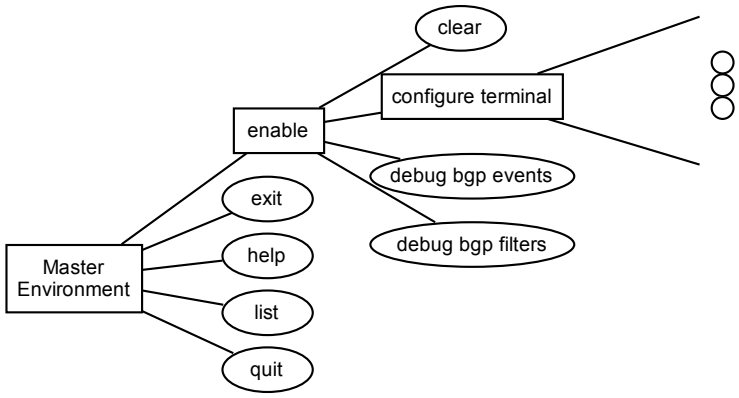

Figure 6.7: Small part of the bgpds VTY tree.

This tree is defined in *bgp_vty.c*. There, both nodes and leaves are defined. Each command is inserted into its environment. This is done using the function *install_element(Node, command)*. A command is defined by a tuple of (name, help message string, parameters and explanation) followed by the code to execute when invoked.*figs. 6.8 and 6.9* how a command is defined and installed.

```
DEFUN (show_ip_bgp_ipv4_paths,
       show_ip_bgp_ipv4_paths_cmd,
       "show ip bgp ipv4 (unicast|multicast) paths",
       SHOW_STR
       IP_STR
       BGP_STR
       "Address family\n"
       "Address Family modifier\n"
       "Address Family modifier\n"
       "Path information\n")
{
  vty_out (vty, "Address Refcnt Path\r\n");
  aspath_print_all_vty (vty);

  return CMD_SUCCESS;
}
```

Figure 6.8: Defining a command.

```
/* "show ip bgp paths" commands. */
install_element (VIEW_NODE, &show_ip_bgp_paths_cmd);
install_element (VIEW_NODE, &show_ip_bgp_ipv4_paths_cmd);
install_element (ENABLE_NODE, &show_ip_bgp_paths_cmd);
install_element (ENABLE_NODE, &show_ip_bgp_ipv4_paths_cmd);
```

Figure 6.9: Installing the command in a node.

6.4 Modifications to the code of bgpd

6.4.1 Implementing ghost-flushing over GNU Zebra

When the SSFNet simulator was modified, we had to modify the decision process to implement the Ghost-Flushing rule. So here, we did the same, the "strike target" was the decision process: **bgp_process()** at **bgp_route.c**.

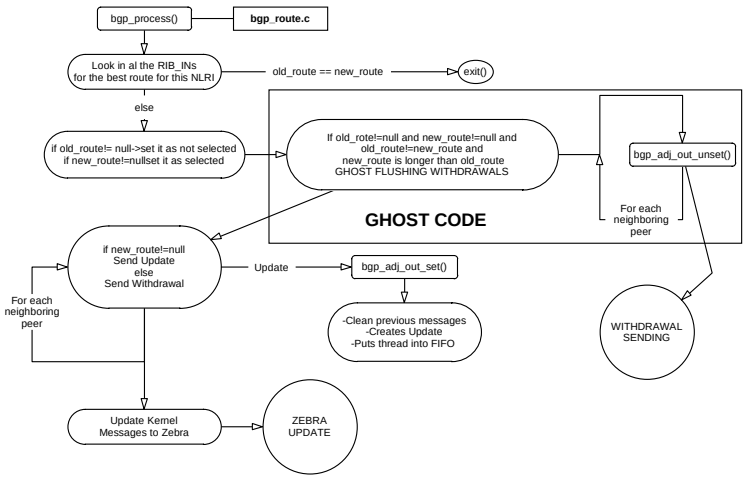
bgp_process()
bgp_route.c
Look in al the RIB_INs for the best route for this NLRI
old_route == new_route
exit()
else
if old_route!= null->set it as not selected if new_route!=nullset it as selected
If old_rote!=null and new_route!=null and old_route!=new_route and new_route is longer than old_route GHOST FLUSHING WITHDRAWALS
bgp_adj_out_unset()
For each neighboring peer
GHOST CODE
if new_route!=null Send Update else Send Withdrawal
Update
bgp_adj_out_set()
For each neighboring peer
-Clean previous messages -Creates Update -Puts thread into FIFO
WITHDRAWAL SENDING
Update Kernel Messages to Zebra
ZEBRA UPDATE

Figure 6.10: Inserting the Ghost-Flushing code in the decission process.

Here we can see the C code that does the work as it is inserted inside the bgp_process function (decisions process) in bgp_route.c:

```
//GHOSTMODIF
// We put it here... the other one is withdrawal anyway
// Conditions:
// GF:activated, we substitue a route, the new one is longer and
//  MRAI is not expired.
mi_thread=peer->t_routeadv;
if (bgp->ghost_flushing && old_select && new_select && old_select != new_select &&
   new_select->attr->aspath->count > old_select->attr->aspath->count &&
  (((int) thread_timer_remain_second(mi_thread))>1))
{
  if (bm->ghost_debug)
  {
     zlog_info ("%s GF-Route Changed for a longer one and MRAI Timer(%d) Not expired.",
         peer->host,(int) thread_timer_remain_second(mi_thread));
     zlog_info ("%s GF-Sending withdrawal for: %s/%d", peer->host,
          inet_ntop (rn->p.family, &(rn->p.u.prefix), buf, BUFSIZ),
     rn->p.prefixlen);
   }
   bgp_adj_out_unset (rn, peer, p, afi, safi);
}
//GHOSTMODIF-END
```

6.4.2 Modifying the VTY configuration environment

Modifying the bgp protocol is not enough, it has to be possible to **enable and disable** the new modifications., also and option to show the debugging information has to be added. To do this we have to

do a small modification to the VTY code, as we showed in previous chapters, we just have to define the new commands and "install" them in the corresponding node.

First we will define two new commands in **bgp_vty.c**: **one to activate (bgp ghost-flushing) the ghost-flushing** rule and another to **deactivate (no bgp ghost-flushing**) it:

```
//GHOSTMODIF
DEFUN (bgp_router_GF,
       bgp_router_GF_cmd,
       "bgp ghost-flushing",
       BGP_STR
       "Enables Ghost Flushing Technique\n")
{
  char ret;
  struct bgp *bgp;
    bgp = vty->index;
  vty_out (vty, "%% Ghost Flushing Enabled.%s", VTY_NEWLINE);
    bgp->ghost_flushing= 1;
  return CMD_SUCCESS;
}

DEFUN (no_bgp_router_GF,
       no_bgp_router_GF_cmd,
       "no bgp ghost-flushing",
       BGP_STR
       "Disables Ghost Flushing Technique\n")
{
  char ret;
  struct bgp *bgp;
    bgp = vty->index;
  vty_out (vty, "%% Ghost Flushing Disabled.%s", VTY_NEWLINE);
    bgp->ghost_flushing = 0;
  return CMD_SUCCESS;
}
//GHOSTMODIF-END
```

Then, we have to **add this two new commands** to the code that installs the commands in the VTY environment. The node we want to install it is in the router node inside the bgp configuration.

```
//GHOSTMODIF
/* GHOSTFLUSHING COMMANDS */
 install_element (BGP_NODE, &bgp_router_GF_cmd);
 install_element (BGP_NODE, &no_bgp_router_GF_cmd);
//GHOSTMODIF-END
```

Now, we have to add two new more commands to **activate and deactivate the debugging messages** associated to the ghost-flushing rule. This time the modifications have to be done in the **bgp_debug.c** file. Here, all the vty commands associated with debugging are coded. These are the new two commands, **(debug bgp ghost-fluhsing, no debug bgp ghost-flushing, undebug bgp ghost-flushing)**:

```
// GHOSTMODIF
DEFUN (debug_bgp_gf,
       debug_bgp_gf_cmd,
       "debug bgp ghost-flushing",
       DEBUG_STR
       BGP_STR
       "BGP Ghost Flushing messages.\n")
{
```

```
    bm->ghost_debug=1;
    vty_out (vty, "BGP Ghost Flushing debugging enabled%s", VTY_NEWLINE);
  return CMD_SUCCESS;
}

DEFUN (no_debug_bgp_gf,
       no_debug_bgp_gf_cmd,
       "no debug bgp ghost-flushing",
       DEBUG_STR
       BGP_STR
       "BGP Ghost Flushing messages\n")
{
    bm->ghost_debug=0;
    vty_out (vty, "BGP Ghost Flushing debugging disabled%s", VTY_NEWLINE);
  return CMD_SUCCESS;
}

ALIAS (no_debug_bgp_gf,
       undebug_bgp_gf_cmd,
       "undebug bgp ghost-flushing",
       UNDEBUG_STR
       DEBUG_STR
       BGP_STR
       "Ghost Flushing messages\n")
// GHOSTMODIF-END
```

Now, again we have to **alter the function installing the nodes** so the commands are installed in the proper nodes:

```
// GHOSTMODIF
  install_element (ENABLE_NODE, &debug_bgp_gf_cmd);
  install_element (CONFIG_NODE, &debug_bgp_gf_cmd);
  install_element (ENABLE_NODE, &no_debug_bgp_gf_cmd);
  install_element (CONFIG_NODE, &no_debug_bgp_gf_cmd);
  install_element (ENABLE_NODE, &undebug_bgp_gf_cmd);
// GHOSTMODIF-END
```

Looking to *fig. 6.11* we can see where the commands have been placed and how to reach them to activate the ghost-flushing rule.

When we are witting **the configuration file for the bgpd daemon**, to activate the ghost-flushing we just have to add *bgp ghost-flushing* after the router command. To activate the debugging, just *debug bgp ghost-flushing*. Adding "no" at the beginning of any of this commands will cancel their effect.

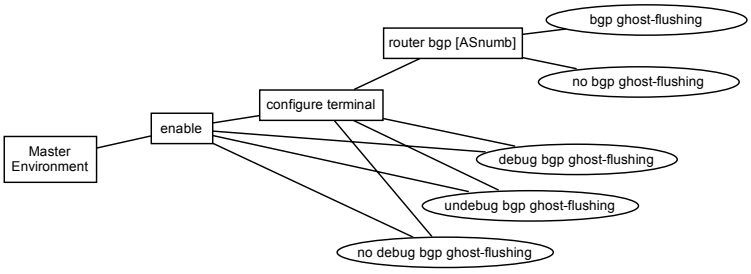

Figure 6.11: This is the VTY node tree with the new commands.

6.5 Tests

6.5.1 Testing and Developing Environment

Setting a proper environment to work with the Zebra daemon, modify it and test it afterwards was really important thing. We considered that spending some extra time in this issue would save time later while test running.

Computers and Network equipment

To do this part of the project, a pool of seven computers and a ethernet switch were assigned in the basement. These are the specs (General) of the computers:

- Processor: 80x86 based, ranging from Pentium to Pentium II.
- Memory: Ranging from 64 to 128 Mb.
- Network Adapters: All of them had two ethernet network cards.
- Operating System: FreeBSD 5.2

All of them have the **same operating system** because we wanted to have only **one compiled copy of the bgpd** to be execute from all the machines. To recompile the code in all the machines was just not an option. From now on, we will refer to them as bgp1, bgp2, bgp3, bgp4, bgp5, bgp6 and bpg7.[1]

The switch was a *hp procurve switch 2524*. Some of the main features relevant to this project were:

- 9.6 Gbps switch fabric integrated on-chip.
- HP Auto-MDIX
- Stacking capability
- VLAN support and tagging: support up to 30 port-based VLANs.
- Web and telnet terminal configuration environments.

Network Layout

The configuration that we established had the a double intention, first, we wanted everything connected to the internet so everything could be **managed remotely** via telnet or ssh. The second is that all the **BGP4 traffic should be in a isolated network** so there wouldn't be any interference from/to external entities.

So this is what we did, we defined two VLANs in the switch. VLAN 1, connected one of the network cards of each computer to the university network, so all the machines would be accessible. VLAN 2, connected all the second network cards. There was no connection between the two LANs. Then, we had to configure all the network interfaces. For all the interfaces connected to LAN 1 we activated the DCHP client so they would automatically get an Internet IP and gateway configuration. For the interfaces connected to the VLAN 2 we assigned manually private IPs in the range 192.168.0.1-192.168.0.7, numbering all the computers from bgp1 to bgp7.

This way we could control all the computers from a remote location (Office), use the switch management tool to simulate link failures for the BGP connections (VLAN2) but at the same time not loosing the remote connection to any of the machines. In *fig. 6.12* we can see a small schema of the network layout.

Exporting files

Another concern was to **centralize the files from GNU Zebra and the configuration scripts** that would run on each machine. The working environment would be automatically set even if hardware faults happened and the machines rebooted (Something that actually happened in bgp3, all the time). In *fig.6.13* we see a small schema of how the files are exported between machines.

[1] In the appendix F pictures of the lab and the switch can be found.

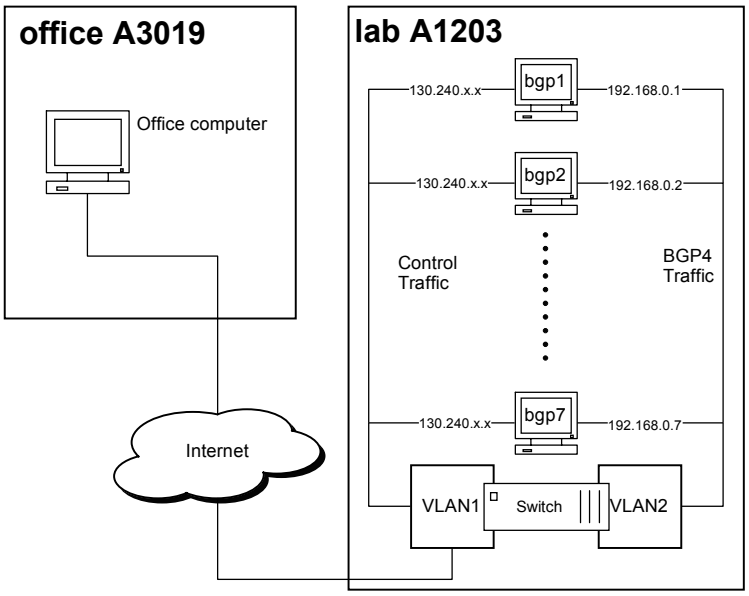

Figure 6.12: Two LANs, one for the BGP4 traffic y the control Traffic.

The system was set in this way:

Zebra files were stored in a University managed remote file system. This allowed us not to worry about security copies or hard-drive faults that would mean critical data lose. Since our machines were administrated by ourselves we could not mount NFS exports from this server (University policy). However, It was also exported using SAMBA, so we remotely mounted this files in every computer using SAMBA clients.

Configuration files All the scripts needed to set up the samba mounting were stored in one machine, bgp1. This one would export them by NFS to all the other machines.

Compiling was always done by bgp1 (The most powerfull of the machines). All the files, source and binaries, were stored in the remote SAMBA file system.

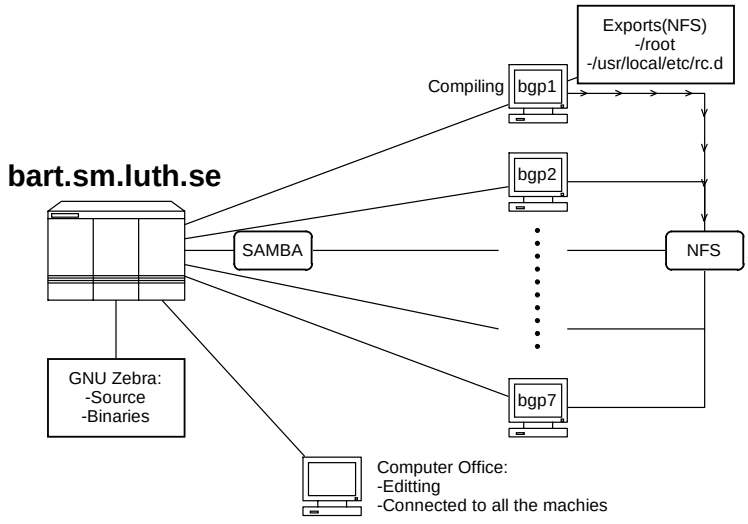

Figure 6.13: Exporting files.

6.5.2 Testing means Programming

To save time during the creation of the configuration files for all the nodes, some small programs were written. We defined a file format that would allow us to define all the bgp configuration for all the nodes. Then, a program would read the file and generate the configuration files for all the machines. Then, the script that starts the bgpd would identify which machine was running the script and feed bgpd the corresponding to the bgpd.

Configuration File Format Here we can see the file format:

```
Number of nodes(n)
[Triangular half square matrix indicating the BGP connection between the nodes. Size n.]
N entries with: [IP of the BGP4 interface] [GF if ghost-flushing activated]
N entries with: [Number of networks exported]#[Network address]#[Network address]
```

An example of a 5 nodes clique, all the nodes with ghost-flushing. Node 5 exports the network IP range 213.98.0.0/16:

```
5
0
1 0
1 1 0
1 1 1 0
1 1 1 1 0
192.168.0.1 GF
192.168.0.2 GF
192.168.0.3 GF
192.168.0.4 GF
192.168.0.5 GF
0#10.0.0.1/24
0
0
0
1#213.98.0.0/16
```

GenConfZ.tcl

1. Function: Creates the configuration files for the nodes. The name for the files will be: bgpd.conf.[number of node].
2. Parameters: Name of the topology file to be read.

Arranca.sh/Para.sh The first script identifies which machine has it been executed on and starts bgpd with parameters -f and the configuration file corresponding to this machine. Para.sh kills all the instances of bgpd running.

6.5.3 Tests and results

During the testing we intended two objectives:

- Test if the ghost-flushing implementation done on Zebra behave as it was supposed to. (Correctness)
- Do small performance tests in clique topologies to see the impact of the ghost-flushing rule in a real routing software.

Correctness of the Implementation

To do this we set up a 5 nodes clique where number 5 was the only one advertising an IP range. We activate the logging option for the ghost-flushing and message sending. We start up all the nodes, wait until the set up process finishes, and then, from the switch management tool, we virtual unplugged the bgp network interface from node 5.

After this, we checked the log files. Checking the logs we try to find the log messages and the withdrawals proper of triggering the ghost-flushing rule. In the case the MRAI timer was expired the corresponding withdrawal message had to be sent.

After some tests the correctness of the implementation was accepted.

System Performance

In this situation we tested two scenarios: 5 nodes and 7 nodes clique. We repeated the tests in the previous section and then measured how long did it take until the network stabilized.

- 5 Nodes:
  - E_{down} with ghost-flushing: 1-3 seconds.
  - E_{down} without ghost-flushing: 50-60 seconds.
- 7 Nodes:
  - E_{down} with ghost-flushing: 2-4 seconds.
  - E_{down} without ghost-flushing: 110-120 seconds.

Looking at the logs we see again (Both with and without the ghost-flushing rule) the convergence behavior described in the simulation chapter. Bigger tests with different topologies could be done, but more nodes (computers) would be needed.

6.6 Conclusions

6.6.1 General Conclusions

In the first place, we could get the odd convergence behavior out of the simulation environment and reproduce it in the real world. This shows that the problem studied during this project exists out of the theory.

About the capacity of **GNU Zebra** as a routing software, our experience working with the software (From the inside to the outside) shows this software as a reliable, inexpensive option to set routers running over conventional hardware. Also, this opinion is supported by articles from the routing community [34] and our experience with the Zebra mailing list [4], showing that the number of users of this software is big and growing. Also the possibility to customize this software gives extra points to it.

About the **ghost-flushing** rule, it shows the same properties observed on the simulation. We can state that our customized version of GNU Zebra implements the ghost-flushing rule as defined on [13]. The results show that it works as expected and with the expected performance.

6.6.2 Odd things about GNU Zebra

There are some characteristics about GNU Zebra that came into our attention. First, it implements the **split-horizon** rule for the BGP4 by default, being hard-coded and impossible to deactivate.

Also, it didn't applies **jitter** to the timers, any timer. There are even comments inside the code indicating it as future work. Finally, there is a strange behavior about how the *MRAI timer* works. It never stops, the timer is initialized to the value of 30 seconds, and when it expires, instead of remaining still it starts again from 30. This seems to be a way of compensating the lack of jittering or just heirloom from earlier versions.

Because of this features we did a special round of simulations, the one without jitter and with split-horizon (A situation that supposedly didn't often happen in the routing softwares).

Chapter 7
Conclusions and Discussions

7.1 Quality of the Simulation Results

How good are the results we got from the simulations done over SSFNet? Well, if we look to the characteristics of the simulator, it tries to simulate everything from the routing protocol to all the layers below. Supposing everything is well implemented and that the simulating engine (That is used for other things that networking) works properly, we can trust the results.

At the beginning of this work, we tried to do the only validation tests we could. In the first place, the BGP4 module comes with a pool of validation tests. This tests use the simulator and compare the results with what they are supposed to be. Actually, when the implementation of ghost-flushing and ghost-buster was finished, we used this validations tests to see if they would affect the BGP4 normal functioning.

However, we also run our own tests. They consisted in trying to reproduce the odd convergence behavior described in [13]. It worked to discard ns-2, so we used it again with SSFNet and it passed it.

Other sources of "trust" on the simulator may come from the opinion on the routing community, and it looks that, at least for the external part of BGP, it works properly, so we can "trust" the results.

Only note to add is that the simulator begins to be unstable when we simulate bigger models than 500 bgp nodes. This problem is due to a faulty implementation of some of the classes defined in the Java virtual machine version we used.

7.2 The Ghost Flushing Rule

Well, as the simulations and the later tests run on Zebra show, it's quite effective. It has many good points:

- **Low CPU consuming**, there are no special mathematical operations or complicated algorithms that need a lot of CPU on the router.
- **Low memory usage**. Actually, no extra memory usage is needed. No extra information is needed to be stored. On the case of the ghost-buster this was a problem. We needed to set timers each update received, just in case that update information could be later chosen to be advertised again. This meant extra memory and cpu, something ghost-flushing doesn't need.
- **No extra information** on the messages or incompatibility with the traditional BGP4. There are no bad side effects due to combining ghost-flushers and old BGP4.
- **General good performance** as seen on the simulation chapter.
- **Good deployment** Looking at the results of mixing normal and ghost-flusher nodes, we notice improvement with a low number of active "flushers".

On the whole, we can say that ghost-flushing rule is a great and easy modification for BGP4. From our humble position, we would recommend router vendors to add to their software as we did with GNU Zebra.

7.3 The GNU Zebra Routing Software

Is GNU Zebra and option as a real routing software? Well, if we look to our experience during this project, the opinion of some "experts"[34] and the community of users, we can say yes, if you need a cheap router. It implements all the main RFCs related with BGP4 and the most important internal routing protocols. It's modular features makes it **reliable** and easy to diagnose in case of fault. Also, the possibility to modify and customize the behavior of the software, makes it appealing in research environments where testing new features can be interesting. They only problem it suffers from is that even if the software is nice, **It won't ever beat an embebed router**: a combination of hardware and software all created to behave as a router will always over perform a conventional hardware with a software running on it. However, it's cheap.

About the ghost-flushing implementation, it was easy to implement (Not to know where, or how: mapping the code) and it seems to work. The tests we drove showed a good performance about it.

One extra comment about Zebra is about the documentation related to it. There is a lot of documentation about the general architecture but not about how the protocols were implemented. It's relatively easy to develop new modules for Zebra. However, to look inside of the code (C code), to analyze how the functions of each protocol are done can be quite hard. A documentation about this work would be needed for contributors to be able to do real work about Zebra. Fortunately, the Quagga project tries to do this. Parts of this document try to be a small map of how some of the BGP4 functions are implemented in bgpd.

Chapter 8
Future Work

8.1 Interaction of Ghost-Flushing with other features

BGP4 has many features that can interact with the ghost-flushing rule. We took into account some of them (Split-horizon, jitter) but not all.

In the first place, a further working line would be to do a **theoretical analysis of the interaction between the ghost-flushing and the split-horizon**. Maybe to try to combine them in one rule, sine both seem to work well together, and create a even more powerful rule.

What about the rest of BGP4? For example, what impact would have the **filter policies** [12] in the performance of the ghost-flushing rule? This and other interactions would be needed to be studied, although probably conclusions about it could be generated during the deployment of the rule on the Internet.

8.2 Testing Zebra to its limits

To test properly the implementation of ghost-flushing and the capabilities of Zebra, we would have needed more computers to create bigger and more complex BGP4 networks.

However, by the time this project was finishing we learnt about a developing test environment thought for this case. The "Universidad Politécnica de Madrid" is working in a virtual environment generator to test network applications. It is called VNUML (Virtual Network User Mode Linux)[17]. It is a system to simulate multiple independent network scenarios on the same machines. It is possible to run a high number of applications emulating they are on different machines.

One possible future work would be to test Zebra and the ghost-flushing using this system, making possible to test systems with tens of nodes.

Parte II

Resumen en Español

Capítulo 1
Introducción

En el principio la redes existían pero estaban aisladas, nada las unía. Sin embargo, la necesidad de intercambiar información entre computadores conectados a redes diferentes apareció. En aquel entonces, los routers fueron inventados para hacer esto posible. El tiempo pasó, y las redes crecieron más y más, hasta el momento en el que nuevos problemas comenzaron a aparecer. Configurar manualmente todos los routers para que supieran alcanzar todas las redes se convirtió en algo demasiado complicado. La solución, escribir nuevos protocolos (Protocolos de enrutado) para que estos routers pudieran comunicarse e intercambiar información de conectividad de manera automática.

Sin embargo, el tamaño de Internet (Todas las redes conectadas) no paró de crecer, llegando el momento en el que aquellos protocolos de encaminado ya no fueron adecuados para las nuevas circunstancias. Como solución, apareció una nueva idea: los Sistemas Autónomos (AS en inglés). Internet fue dividida en Sistemas Autónomos (AS) y dos tipos de protocolos fueron escritos: externos e internos. Los protocolos de encaminamiento internos eran utilizados para intercambiar rutas dentro de los ASs y consistían en versiones de los protocolos de enrutado utilizados anteriormente. Los protocolos exteriores, tenían una misión diferente: intercambiar rutas entre los routers que estaban en los bordes de ASs diferentes. Eran los 80 y el momento en el que Border Gateway Protocol (BGP) fue inventado: el protocolo que se convertiría en un estándar en Internet.

De nuevo, tras algún tiempo, nuevos problemas aparecieron. Como consecuencia, nuevas versiones de BGP fueron desarrolladas. A principios de los 90, la última versión, BGP4, fue publica e implantada en Internet. BGP4 se ha convertido en "el pegamento que mantiene a Internet junta", por lo tanto, cualquier modificación que pueda mejorar su comportamiento tendrá un efecto positivo en toda la Internet. Este proyecto trata sobre una serie de propuestas que pretenden mejorar el rendimiento del BGP4 en ciertas situaciones.

Esta parte de la memoria pretende ser un pequeño resumen de la memoria completa. Aunque incluye parte de las ideas más importantes de este proyecto, no podemos dejar de recomendar la lectura de toda la memoria para alcanzar una visión completa y detallada del trabajo desarrollado.

1.1 Motivación, propósito y objetivos

BGP4 es el protocolo de encaminamiento externo (EGP) más utilizado en estos momentos. Su comportamiento es crucial para la estabilidad de Internet haciendo que cualquier defecto o problema que pueda sufrir tenga un impacto muy serio en las comunicaciones mundiales [12, 13, 15]. De hecho, algunas problemas han sido descritos relacionados con el rendimiento de BGP4, afectando el tiempo de respuesta y retrasos en las comunicaciones a través de Internet. Esto lleva a que servicios de tiempo real, como el de video-conferencia o voz sobre IP y que actualmente se intentan implementar sobre Internet, no puedan competir con servicios sobre la fiable, rápida y de baja latencia red telefónica digital.

Como algunos estudios han descrito, el rendimiento de BGP4 puede disminuir dependiendo de la topología de red [12]. En casos en los cuales partes de la red están "caídas", podría costar incluso minutos hasta que el resto de los nodos se reconfigurasen según la nueva situación de red. Es es llamado el *problema de convergencia: Tiempo que cuesta hasta que la red se establece después de que un evento haya alterado su situación.*

Muchas soluciones ha sido propuestas pero, entre ellas, hemos elegido estudiar las técnicas descritas en Bremler-Barr et al. [13]. Analizar su base teórica, simular BGP4 aplicando las propuestas y, dependiendo del resultado, implementarlas sobre un software de encaminamiento son los objetivos de este proyecto.

1.2 Estructura y método de este trabajo

BGP4 es un protocolo crítico para Internet, previamente a aplicar cualquier modificación al estándar, una serie de pasos deben ser llevados. Una modificación erronea o inconsistente del protocolo podría llevar a un inadmisible fallo en masa. Este trabajo es el estudio/análisis de una serie de propuestas para mejorar BGP4, por lo tanto, quisimos darle una estructura que permitiese que el resultado fuese todo lo coherente y correcta posible. Cada paso implica que el anterior valida el trabajo ya hecho. Dichos pasos han sido:

1. Estudio del Protocolo BGP4 tradicional y su entorno.
2. Análisis del problema y la teoría detrás de las soluciones.
3. Simulación de las version tradicional junto con las modificadas de BGP4 comparando los resultados obtenidos.
4. Implementación de las soluciones validadas en un software real de encaminamiento: GNU Zebra.

Capítulo 2

Protocolo de encaminamiento BGP4 y su entorno

2.1 Descripción del Protocolo BGP4 Protocol

BGP4 es la versión estándar de el protocolo de encaminamiento externo más usado en Internet. La primera versión fue publicada en Junio de 1989 en el RFC-1105, aunque la última versión es BGP4 y está descrita en RFC-1771 [28] y es la estudiada en este trabajo. Como se verá a lo largo de este capítulo, BGP4 implementa técnicas como detección de ciclos mediante vector de ruta, CIDR o políticas de propagación que le permiten funcionar correctamente (O casi) en Internet.

2.1.1 Vectores de Ruta y Detección de ciclos

El enfoque de BGP4 hacia la idea del vector de ruta es bastante radical. Cada mensaje de actualización sobre una ruta incluye una lista de todas las ASs a través de las cuales la información ha pasado. Cuando un router quiere retransmitir información que ha recibido, incluirá su número de AS al principio de esa lista de ASs, de este modo es como el vector de ruta de va construyendo. Esta lista de ASs es conocido como ***Camino de ASs***.

Este vector de ruta tiene una función muy clara, permitir la **detección de ciclos**. Uno de los principales problemas de otros protocolos de encaminamiento son los ciclos en la topología de red (Por ejemplo el RIP). Sin embargo, con el camino de ASs un router puede comprobar si su número de AS está en la lista y,en caso positivo, descartar tal ruta. En caso negativo, puede aceptar la ruta y retransmitirla añadiendo su número de AS al principio del camino de Ass.

Esta técnica también tiene inconvenientes. El principal es que la lista de ASs atravesadas por una ruta es almacenada en la memoria de los routers y enviada en los mensajes de actualización. Considerando la cantidad de redes que existen esto puede significar un gran coste en memoria y sobrecarga de mensajes. Algunos estudios [20] afirman que con BGP3, para 100.000 redes, una longitud media por camino de 20 ASs y unas 3.000 ASs, el coste en ancho de banda de transmitir una tabla completa de encaminado sería de unos 500.000 bytes. Si miramos a como las rutas son almacenadas en los routers, esta cifra es sobradamente excedida fácilmente. Sin embargo, este problema sería solucionado aplicado CIDR y agregación de rutas en BGP4.

2.1.2 Atributos de ruta

Los mensajes de actualización portan la información encaminamiento como atributos. Estos atributos pueden ser clasificados según dos criterios: Transitivos/no transitivos (si un atributo debe ser incluido en caso de que un ruta sea retransmitida) y obligatorios/opcionales (Obligatorios son aquellos que todos los routers deben entender y obligatorios son aquellos implementados en versiones del protocolo que no son estándar).

Los principales atributos obligatorios:

Camino de ASs Lista de ASs atravesadas por la ruta.

Origen si la ruta es interna a la AS o si ha sido recibida desde otra AS.

Inalcanzable , si activa, significa que el destino sobre el que este mensaje trata no puede ser alcanzado.

Métrica entre ASs .

Siguiente Nodo el nodo donde empieza la ruta hasta el destino.

2.1.3 El Protocolo

Este protocolo utiliza conexiones TCP entre los nodos BGP4 para enviar los mensajes asociados. El puerto utilizado por defecto es el 169. La razón para utilizar TCP es simplificar todas las funciones de control de errores o envío de paquetes que complicarían en demasía BGP. Esta decisión fue inicialmente criticada por la poca consistencia de TCP en situaciones de congestión de red. Sin embargo, las nuevas implementaciones de TCP incluyen funcionalidades que corrigen este problema.

Los mensajes enviados por este protocolo son de cuatro clases:**inicialización, actualización, notificación y keepalive**. Las funciones de los mensajes son las siguientes:

Inicialización : estos mensajes son usados en el establecimiento de sesión entre dos nodos. Es el primer mensaje enviado en dicha sesión, e incluye información como:

- Número de AS del router que envía el mensaje.
- Tiempo de hold utilizado para la función de "keep alive".
- Un identificador: la IP del interfaz desde el que se establece la comunicación.
- Información de autenticación.
- Versión de BGP utilizada.

Para aceptar este mensaje y dar la sesión por establecida, el otro nodo solo ha de mandar un mensajes de keepalive.

Actualización , una vez que la sesión está establecida la información de encaminamiento es intercambiada usando los mensajes de actualización. Un mensaje de actualización solamente contiene información sobre un destino. Dicha información incluye:

- Redes alcanzables por este camino, también conocido como NLRI(Network Layer Recheability Information).
- Camino de ASs, ASs atravesadas por esta información de encaminamiento.
- Origen de la información.
- "Vecino" que está mandando este mensaje.
- Métricas entre ASs, información utilizada en parte de la elección de ruta.
- Inalcanzable, indicando que los destinos indicados ya no son alcanzables.

Cuando un mensajes de actualización es recibido, la información es procesada por el protocolo. Primero, se comprueba el camino de ASs, si nuestro número de AS ya aparece, entonces existe un ciclo y la ruta es descartada. Si no es descartada y anuncia una nueva ruta, esta ruta es insertada en una tabla llamada RIB-IN, habiendo una RIB-In por cada vecino con el que el router ha establecido una sesión BGP4. Si el mensaje es de "retirada" de una ruta, toda información sobre ese(os) destino(s) en el RIB-In correspondiente al vecino que envía el mensaje es eliminada.

Después de hacer cambios en los RIB-in se ejecuta el "proceso de decisión", este proceso evalúa las rutas que hay en los RIB-INn y si hay una mejor que la que se había elegido hasta ahora (O no existe ninguna) se pone la nueva (O se elimina la vieja) en el LOC-RIB (Solo hay uno). Esta estructura almacenas las rutas elegidas para alcanzar cada destino conocido.

Si el LOC-RIB es modificado, puede existir la posibilidad de que sea necesario enviar nueva información a nuestros vecinos. Para ellos existe otro proceso de decision que elegirá qué debe ser anunciado. Los nuevos anuncios o retiradas son puestas en otra tabla, RIB-OUT. Más tarde, la información esta tabla será procesada y las actualizaciones correspondientes serán enviadas.

Keep-alive son utilizadas por los nodos BGP para informar a sus vecinos que siguen funcionando y no hay ningún problema. Si un nodo no recibe un mensaje de keep-alive en más del tiempo de hold negociado en el establecimiento de sesión, intentará saber si el nodo remoto funciona mediante mensajes de notificación.

Notificaciones son utilizadas para manejar los posibles errores que puedan ocurrir alrededor de BGP.

Hay otras características y normas que existen alrededor de los mensajes, entre las más importante están:

Temporizador MRAI timer or Minimum Router Advertisement Interval. Esta es una limitación para controlar la sobrecarga y comportamiento de BGP4. En la configuración estándar,*per peer*, un nodo no puede mandar más de un mensaje de actualización con nueva información de ruta (Anuncio) por tiempo de MRAI. El valor recomendado para este temporizador es de 30 segundos. Puede también configurarse *per destination*. En tal caso, la limitación es no mandar más de un mensaje sobre una NLRI en concreto a un vecino. El valor recomendado en el RFC para este temporizador es de 30 segundos.

Jitter Todos los timers, keep-alive, MRAI o inicialización, deben aplicar un jitter aleatorio para evitar problemas por sincronización. i.e. colisión de los mensajes de apertura u otras situaciones debidas a la disposición de red.

Procesos de decisión en el RFC-1771 [28], tres diferentes procesos de decisión son especificados. El primero evalúa los mensajes de actualización y modifica si es necesario los RIB-INs. El segundo proceso de decisión revisa los RIB-INs y modifica el LOC-RIB si es necesario. Esto puede ocurrir por tres razones: porque todas las referencias a un destino han sido eliminados de los RIB-INs, por lo que la ruta en el LOC-RIB ya no es valida y debe ser eliminada. Otra causa es que ahora hay una nueva ruta mejor en los RIB-INs que sustituye a la existente en el LOC-RIB. Finalmente puede ser que un vecino haga un segundo anuncio sobre una mismo destino (Retirada implícita), en tal caso la nueva ruta puede ser colocada en el LOC-RIB (Si es mejor que otras que teníamos). Todos estos procesos se ejecutan en exclusión mutua.

Retirada implícita , si un nodo cambia el camino que usaba para llegar a un destino, lo había anunciado antes pero el temporizador MRAI no está expirado, en vez de enviar un mensaje de retirada, esperará hasta que dicho temporizador expire y enviar un mensaje de anuncio con el nuevo camino. Esto se llama retirada implícita.

Elegir la mejor ruta para cada destino se hace en el segundo proceso de destino. Depende de la implementación del protocolo y las políticas de la institución que controla la AS. Ejemplos de los criterios a seguir a la hora de elegir un camino (Por orden de importancia):

- Longitud del camino de ASs.
- Métrica de los diferentes caminos.
- Preferencia local. Si hay diferentes maneras de "entrar" en una AS, esta AS puede preferir ser alcanzada por un camino u otro.
- Otras que los administradores quieran implementar.

Las políticas de distribución son también algo importante a tener en cuenta. Incluso aunque diferentes ASs estén físicamente conectadas (red), y hay sesiones entre sus BGP routers no implica que intercambien toda la información de encaminamiento que conocen. Esto es lo que se conoce como relaciones entre ASs [18]. Internet es una red que pertenece a una serie de compañías con intereses económicos, las compañías cobran por transportar tráfico y dar conectividad a otras redes. Esto puede significar que una AS podría permitir un cierto tráfico desde un vecino a un destino, pero no a otro (Quizás porque el vecino ha contratado a la compañía dueña de la AS para alcanzar solo una serie de destinos).

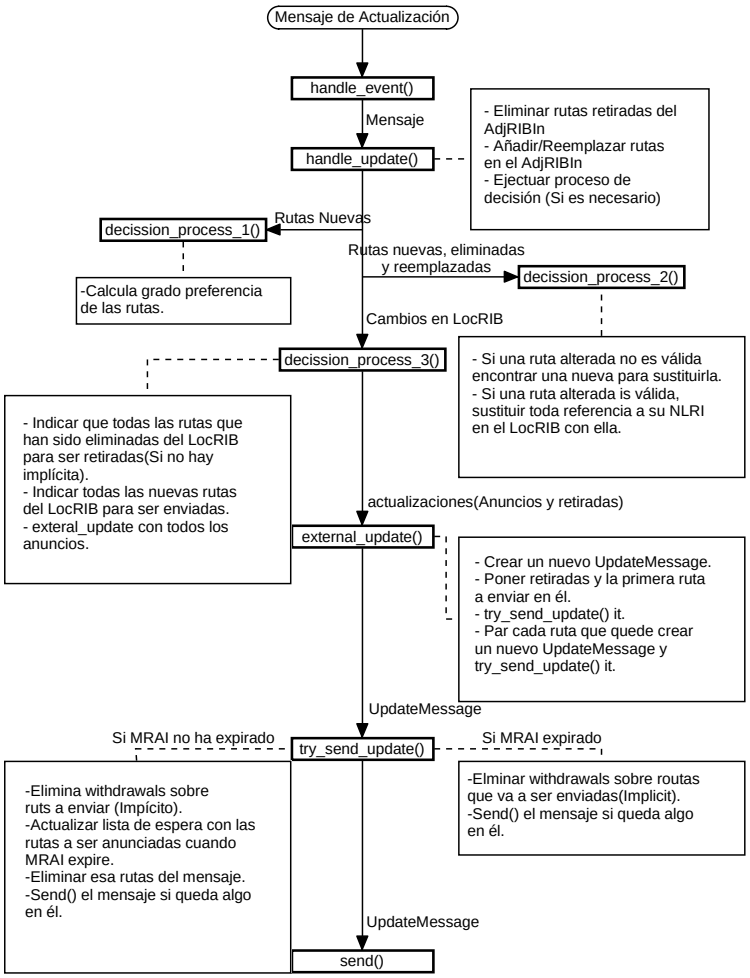
Mensaje de Actualización
handle_event()
Mensaje
handle_update()
- Eliminar rutas retiradas del AdjRIBIn
- Añadir/Reemplazar rutas en el AdjRIBIn
- Ejectuar proceso de decisión (Si es necesario)
Rutas Nuevas
decission_process_1()
-Calcula grado preferencia de las rutas.
Rutas nuevas, eliminadas y reemplazadas
decission_process_2()
Cambios en LocRIB
decission_process_3()
- Si una ruta alterada no es válida encontrar una nueva para sustituirla.
- Si una ruta alterada is válida, sustituir toda referencia a su NLRI en el LocRIB con ella.
- Indicar que todas las rutas que han sido eliminadas del LocRIB para ser retiradas(Si no hay implícita).
- Indicar todas las nuevas rutas del LocRIB para ser enviadas.
- exteral_update con todos los anuncios.
actualizaciones(Anuncios y retiradas)
external_update()
- Crear un nuevo UpdateMessage.
- Poner retiradas y la primera ruta a enviar en él.
- try_send_update() it.
- Par cada ruta que quede crear un nuevo UpdateMessage y try_send_update() it.
UpdateMessage
Si MRAI no ha expirado
try_send_update()
Si MRAI expirado
-Elimina withdrawals sobre ruts a enviar (Impícito).
-Actualizar lista de espera con las rutas a ser anunciadas cuando MRAI expire.
-Eliminar esa rutas del mensaje.
-Send() el mensaje si queda algo en él.
-Elminar withdrawals sobre routas que va a ser enviadas(Implicit).
-Send() el mensaje si queda algo en él.
UpdateMessage
send()

Figura 2.1: Esquema (Simplificado) de el procesado de un anuncio en la implementación de BGP4 en SSFNet.

2.1.4 Soporte a CIDR

Internet se comporta a veces como un ser vivo, intentando crecer lo máximo posible en el menor tiempo posible. Como consecuencia, los posibles problemas de escalabilidad aparecen antes de los esperado y amenazan con llevar la red a un caos total. Este fue el caso del sistema de asignación de IPs a principio de los 90. Cuando TCP/IP fue diseñado en los 80, 32 bits parecían suficientes para todos los equipos conectados Internet, siendo los 8 primeros bits la dirección para la red y los 24 restantes para el equipo. Poco después, las tres clases de redes (A,B,C) fueron añadidos para ajustarse a las diferentes necesidades en tamaños de red. Sin embargo, en los 90, el nuevo sistema empezó a tener problemas. En primer lugar, las redes de tipo B estaban a punto de acabarse y las tablas de encaminamiento estaban creciendo demasiado. A "parche" al diseño del encaminado IP fue aplicado entonces, dando tiempo hasta la solución definitiva: IPv6. Este parche es conocido como "Classless Inter-Domain Routing" (CIDR).

Ocupación de redes Clase B y Crecimiento en las Tablas de encaminamiento

Como podemos ver en la figura 2.5, las redes de clase A eran demasiado grandes y las de clase C demasiado pequeñas, así que las redes de clase B se convirtieron en una solución muy popular para toda clase de organizaciones conectadas a internet (Demasiado popular, de hecho). Llego un momento en el que casi todas las direcciones de red de clase B habías sido asignadas pero se demandaban más. Las dirección de

clase C eran demasiado pequeñas, y solo había 256 redes de tipo A. Un peligro real de sobreocupación de direcciones apareció.

Clase	Bits de Red	Bits de Host	Hosts por Red
A	8	24	16.777.214
B	16	16	65.534
C	24	8	254

Figura 2.2: Distribución de clases IP

Además, como el número de rutas conectadas se incrementaba constantemente, las tablas de encaminado se hicieron más y más grandes dentro de los routers. Esto tenía un doble efecto de incrementar tanto el uso de memoria como la carga en CPU para manejar tablar mayores. Este problema podría afectar al tiempo de respuesta en caso de fallo en Internet.

Classless Addressing, Route Aggregation y BGP4 como un Protocolo de Encaminado entre Dominios

Las direcciones sin clase se componen de una IP completa y una máscara de red. Dicha máscara indica que bits de la IP pertenecen a la dirección red y cuales al host. De este modo, podemos dividir redes A en redes más pequeñas y unificar redes C contiguas para formar otras más grandes. Del mismo modo, podemos definir redes que se ajusten a las necesidades de un cliente que contrate un rango de IPs. De este modo, el problema de sobre ocupación de direcciones IP se había solucionado temporalmente hasta la llegada del IPv6. El intercambio de información de encaminado entre diferentes dominios que no siguen la estructura de clases es el llamado CIDR.

Hasta 1992 no había relación entre las asignación de IPs y la estructura física de Internet. Pero entonces, una nueva estrategia "direccionamiento por proveedor" se impuso. Desde los niveles más altos, los proveedores de red poseían un rango de IPs y sus clientes deberían estar dentro del rango de IPs de sus proveedores. Esto implica una gran ventaja para el encaminado. Comparemos el antiguo escenario con el nuevo. En el antiguo, tenemos dos destinos de rango IP: IP1 e IP2. Ambos se conectan a Internet por medio del mismo proveedor pero sus IPs no tenía nada que ver. En una tabla de encaminado debería haber dos entradas sobre como alcanzarlos. Sin embargo, en el sistema nuevo, si ambos están bajo el mismo rango de IPs, la única entrada necesaria en una tabla de encaminado es al rango del proveedor y ya se encargarán los routers del proveedor de enviar el tráfico al lugar correspondiente. Como se puede ver, un impacto positivo en todos los routers de Internet se da gracias a esta nueva política de "supernetting".

Pero, ¿Qué tiene que ver todo esto con BGP4? BGP3, la versión previa a BGP4, no soportaba CIDR o el "supernetting", esta es la gran diferencia que existe entre ambas versiones. BGP4 es capaz de agregar rutas que comparten partes de su dirección de red. Desde la implementación de BGP4 en Internet, los problemas antes descritos anteriormente fueron aliviados. Además, como BGP4 sabe a través de que ASs una ruta pasa, puede calcular si merece la pena agregar destinos bajo una red mayor. De hecho, existe un atributo definido en el RFC-1771 [28] para indicar si una ruta puede ser agregada o no.

2.1.5 BGP Interno (IBGP)

Aunque este trabajo trata sobre temas relacionados con la parte externa the BGP4, no podemos seguir sin mencionar su papel como protocolo interno. BGP4 puede ser usado entre routers dentro de una AS, sustituyendo otros protocolos como OSPF o RIP (Incluso aunque BGP4 y estos protocolos pueden coexistir). La idea inicial es establecer un red BGP totalmente interconectada entre los routers del borde de la AS y los routers internos. De esta manera las rutas son diseminadas por toda la AS, aunque el sistema parece del todo ineficiente. Otras opciones han sido propuestas, algunos ejemplos son las técnicas de Route Reflector [30] o Confederaciones (También aplicado a BGP externo) [25]. Intentan reducir el número de sesiones establecidas entre routers.

En **Route Reflector**[30], el enfoque se fundamenta en la idea de que podemos dividir una AS en sectores e internamente a los sectores hay una red BGP totalmente interconectada. Las rutas son provistas por *route reflector* para todo el sector. Este route reflector es simplemente otro router BGP4 qué está conectado a los routers del borde (Puede ser un router de borde él mismo) y otros route reflectors.

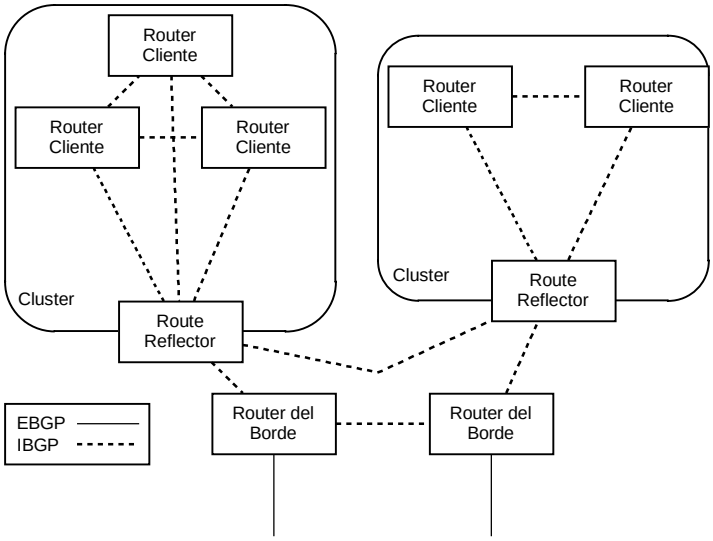

Figura 2.3: Ejemplo de una configuración de Router Reflectors

Otra posibilidad son las **Confederaciones**[25]. La idea es o bien agrupar varias ASs en una AS virtual mayor, o dividir una en otras menores (En caso de BGP externo o interno). De esta manera evitamos las excesivas conexiones de un modelo totalmente interconectado.

2.2 Relaciones entre Sistemas Autónomos

Este trabajo trata sobre los problemas de BGP4 en algunas topologías de red, en este capítulo intentaremos mostrar algunos ejemplos de la relaciones BGP más típicas que se pueden entontar tal y como son descritas en [18]. Este capítulo reseña partes de este artículo importantes para este trabajo. En capítulos posteriores relacionaremos estos ejemplos con la situación problemática.

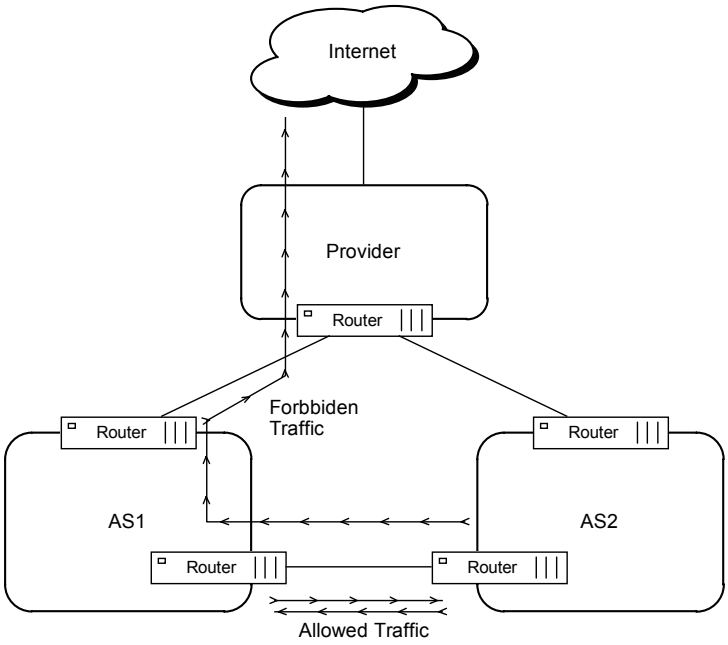

Figura 2.4: Filtros aplicados.

Estas relaciones existen por una serie de razones, en primer lugar la **topología de red**: como las diferentes ASs y redes están físicamente conectadas. El otro factor son las **políticas de intercambio** establecidas entre las ASs debido a relaciones comerciales o económicas entre las instituciones que poseen las ASs. Las políticas de intercambio se manifiestan en los filtros que alteran el intercambio de rutas entre routers de borde. I.e. Dos ASs han contratado acceso a internet a través del mismo proveedor, pero además están conectadas entre sí (*fig. 2.7*). Una de ellas podría querer acceder a la otra pero no permitir que la otra utilice su conexión a Internet. Este un ejemplo de causas comerciales.

2.2.1 Clasificación de Relaciones y Disposición en Internet

Exportando a un proveedor : Intercambiando información con un proveedor, una AS puede exportar sus rutas y las rutas de sus clientes pero normalmente no exporta las de su proveedor o las de sus iguales (peers).

Exportando a un cliente : Intercambiado información de ruta con un cliente, una AS puede exportar sus rutas, las de sus clientes, sus iguales y su proveedor.

Exportando a un igual(peer) : Intercambiando con un igual, una AS puede exportar sus rutas y las de sus clientes, aunque usualmente no exporta las de sus iguales o proveedores.

Exportando a un Hermano : Intercambiando con un hermano, una AS puede exportar sus rutas, las de sus proveedores, clientes e iguales.

Básicamente, Internet es una conjunto de nodos estructurado en forma de árbol-pirámide pero con conexiones entre nodos entre niveles. Unas ASs forman parte de la estructura de ´´back-bone", estas proveen a otras ASs con interconectividad, las cuales proveen de conexión a otras redes menores, y así sucesivamente hasta que llegamos al acceso de los hosts en el "borde" de la red. BGP4 es el protocolo de encaminado externo que intercambia información entre estos niveles (O incluso elementos del mismo nivel). Como podemos ver, se dan escenarios de una alta interconectividad y la relación entre ASs se controla por medio de filtros. El problema a tratar en este trabajo (Como veremos más adelante) ocurre en situaciones de una alta interconectividad.

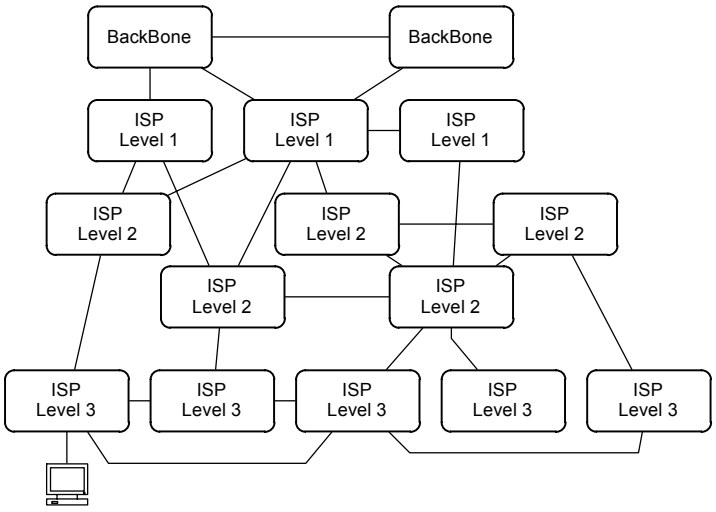

Figura 2.5: Caótico ejemplo de proveedores de Internet.

Capítulo 3

Ghost Flushing y Otras Propuestas

Como se ha afirmado antes, BGP4 es un protocolo que resolvió muchos problemas que ocurrían en Internet. Pero a su vez, sufre de varios problemas derivados de los diferentes entornos (Políticos y topológicos) que debe afrontar. Un estándar demasiado abierto que permite demasiadas implementaciones diferentes [31], inestabilidad de rutas[16, 33], mala interacción entre BGP interno y externo[33] o tiempos de convergencia demasiado altos[12, 13, 15]en algunas situaciones son solo algunos ejemplos de los varios problemas que adolecen BGP4. En este trabajo analizaremos un pequeño subconjunto del último problema indicado, el tiempo de convergencia. Este capítulo definirá el ámbito abarcado por este trabajo y presentará (Y analizará) las soluciones propuestas para solucionar dicho problema.

3.1 El Problema de la Convergencia

3.1.1 Algunas Ideas Iniciales

Antes de entrar dentro del problema, es necesario que definamos una serie de conceptos comunes relaciones con el problema a tratar.

Redes serán vistas como grafos formados por nodos (Routers BGP4) y arcos conectándolos (Sesión BGP4 entre dos routers).

Evento Cualquier cosa que pueda ocurrirle a la red, un arco que aparece o desaparece, o nodos que fallan o comiencen a funcionar. I.e. Fail-down, fail-over, system-up, shorter-path.

Fail-down Un destino deja de ser alcanzable.

Fail-over Un destino es alcanzable por un camino más largo que antes.

System-up Un destino que no estaba disponible ahora es alcanzable.

Shorter-Path Un destino es alcanzable por un camino más corto que antes.

Tiempo de Convergencia Tiempo que transcurre desde que se produce un evento en la red (Fail-down, fail-over, system-up, shorter-path) hasta que la configuración de la red se estabiliza y las tablas de encaminado alcanzan un estado final (No cambiarán hasta que otro evento ocurra).

E_{up} Tiempo de convergencia si un evento system-up ocurre.

E_{down} Tiempo de convergencia si un evento fail-down ocurre.

E_{longer} Tiempo de convergencia si un evento fail-over ocurre.

$E_{shorter}$ Tiempo de convergencia si un evento shorter-path ocurre.

3.1.2 Tiempo de Convergencia Elevado y la Información Fantasma

Como se muestra en algunos artículos [12, 13, 15] hay una fuerte relación entre la topología de red y el tiempo de convergencia. Pero además, **otro factor crítico en el fenómeno del tiempo de convergencia elevado es el uso de temporizadores MRAI**. Después de cualquier cambio en la red, siempre aparece la limitación de que un router BGP no puede mandar más de dos anuncios (Nueva ruta) al mismo router en MRAI segundos. Si son necesarios muchos anuncios, el tiempo derivado de las esperas por el temporizador MRAI se acumula dando lugar a la mayor tiempo necesario para la convergencia.

Las propuestas tratadas en este trabajo y extraídas de *Bremler-Barr et all*[13], están basadas también en otra idea: *Una mentira es la raíz de otras muchas, y en el mundo de las redes esto continua de manera recursiva*. Esto significa que si un nodo emite un anuncio basado en información que ya no es válida (Aunque no lo sepa), la actualización enviada por dicho nodo contendrá información que tampoco es válida. Entonces, otros nodos enviarán actualizaciones basándose en dicha información, y así hasta el final. Después de algún tiempo, la información falsa será eliminada de la red (Flushed), pero costará aún más mensajes, más anuncios (Actualizaciones con nuevas rutas) y mucho más tiempo. Si a todo esto le añadimos la limitación del MRAI, es fácil ver que éste puede ser el origen de una situación de tiempo de convergencia alto.

Después de este ejemplo podemos definir la idea de **Información Fantasma**: contenido de tablas de encaminamiento, suposiciones o mensajes generados a partir de información que ya no es válida. Si seguimos el clásico modelo de BGP4 podemos ver que esta información fantasma engendrará más información fantasma y así hasta el final.

Mostremos un pequeño ejemplo del impacto de la **información fantasma en un escenario fail-down**[13]:

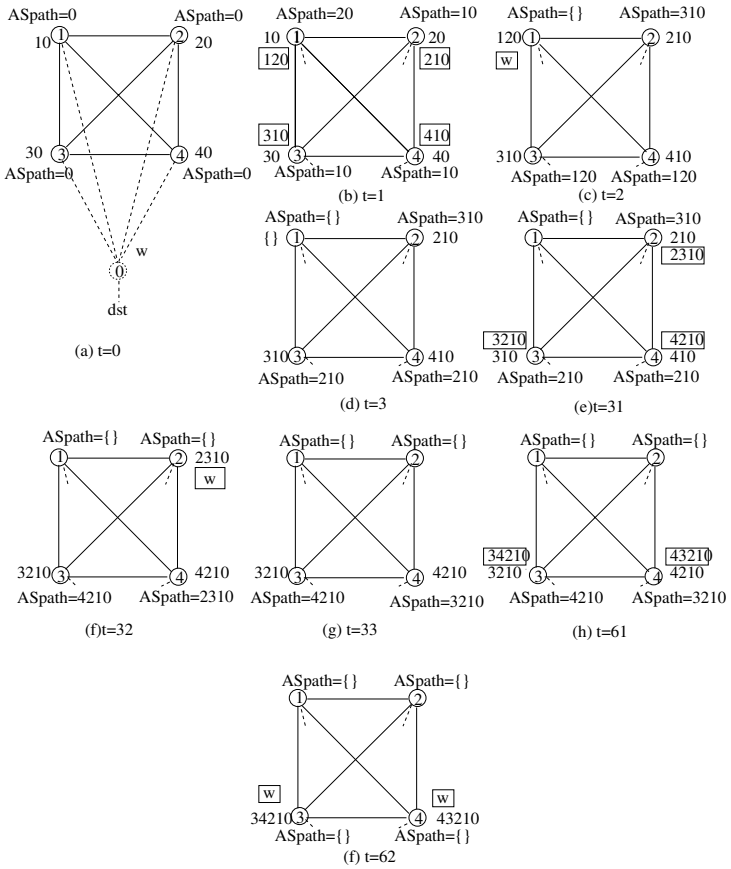

Figura 3.1: Ejemplo de información fantasma creciendo en un escenario de fail-down con un alto E_{down}.

En *fig. 3.1* podemos ver una topología de red clique con 5 nodos totalmente conectados. Para entender la notación, el número dentro del nodo es el número de AS al que pertenece. El número a su

lado es el último camino AS hasta el nodo 0 que ha anunciado. El número dentro del cuadro es el camino AS que será anunciado en el siguiente turno, una "w" significa una mensaje de retirada. El camino AS que está siendo usado en ese momento es el indicador "ASpath=". Asumimos que es una red síncrona en la que los eventos ocurren en turnos de 1 segundo. Si un mensaje ese enviado de un nodo a otro, llegará en el siguiente turno.

Ahora, analizando lo ocurrido en este ejemplo. En t=0, todos los nodos puede alcanzar al nodo 0 directamente, pero entonces este nodo cae. En el siguiente turno todos los nodos eligen un nuevo camino y quieren anunciarlo. Miremos a lo que ocurre a uno de los nodos como ejemplo. En t=1, el nodo 1 cree que otros nodos pueden acceder al nodo 0, por lo que elige al mejor. Como todos tienen un camino de características similares elige el de identificador menor (2). Entonces, el nodo 1 los anunciará como nuevo camino usando 1-2-0 como camino AS. Este es el primer ejemplo de información fantasma nacida de otra información fantasma. El nodo 1 no sabe que el nodo 2 no puede acceder al nodo 0 tampoco, por lo tanto está tomando decisiones fundamentándose en información ya no válida. Pero sigamos observando lo que ocurre después. El resto de los nodos han hecho lo mismo pero eligiendo al nodo 1 como siguiente salto para alcanzar 0. El nodo 1 se dará cuenta de esto en t=2 y enviará inmediatamente un mensaje de retirada a todos los nodos. Sin embargo, es demasiado tarde, en t=3, todos los nodos conocerán que el nodo 1 no puede alcazar a 0 y elegirán otro camino. Pero ya habrán enviado una ruta fantasma un segundo antes, y no podrán enviar un nuevo anuncio hasta que el temporizador MRAI no expire (Casi 30 segundos). Cuando pase ese tiempo, volverán a anunciar una nueva ruta, esta vez eligiendo 2 como próximo salto, es decir más información fantasma. El proceso se repite y la espera de 30 segundos reaparece. Así sucesivamente hasta que toda la información termina por desaparecer.

En este ejemplo se puede apreciar como la combinación de una topología altamente interconectada, la limitación MRAI y la información fantasma tiene como consecuencia un alto tiempo de convergencia: 60 segundos. Un tiempo demasiado alto para algunos servicios de tiempo-real que funcionan sobre la red de redes. Éste es el problema a estudiar por este trabajo, de todas las situaciones de alto tiempo de convergencia las relacionadas cono medios muy interconectados. Intentaremos corregir el problema siguiendo las propuestas del artículos sobre Ghost-Flushing [13].

Otra situación que merece la pena ser estudiada es qué ocurre con **la información fantasma y el caso fail-over**. Miremos el ejemplo, como podemos ver en *fig. 3.2*, la información fantasma no es eliminada del sistema hasta que el nuevo camino atraviesa todos los nodos del camino alternativo. Como podemos ver en este ejemplo, es un tiempo elevado.

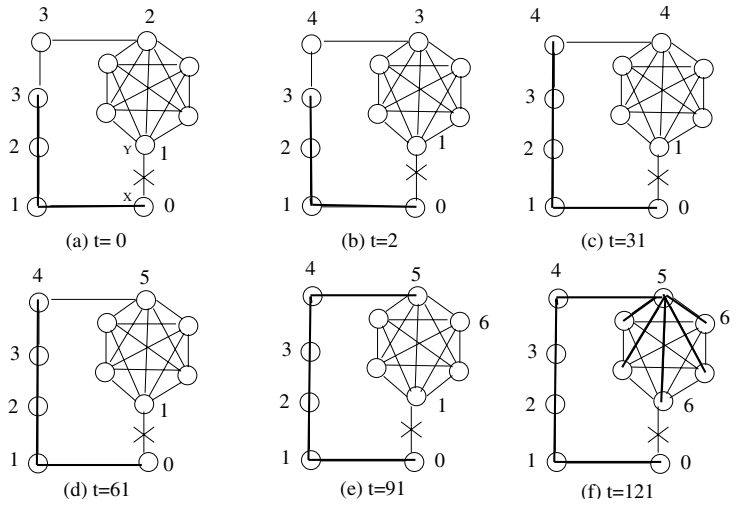

Figura 3.2: Información fantasma extendiéndose en el escenarion fail-over.

3.2 Regla Ghost Flushing

3.2.1 Como Funciona

La idea detrás de la regla ghost-flushing es intentar eliminar la información fantasma lo antes posible. Esta modificación del protocolo BGP4 quiere evitar cualquier alteración de los mensajes utilizados hasta ahora por el protocolo o cualquier medida que implica un alto consumo de CPU/memoria en los routers. ¿Qué podemos hacer entonces?

Identificación de la información fantasma Si cambiamos el camino elegido para alcanzar un cierto NLRI a un camino peor, significa que el camino antiguo ya no es válido, es información fantasma. Pudimos haber anunciado este camino antes, así que hemos creado información ahora fantasma.

Acción a tomar La primera opción es enviar inmediatamente un nuevo anuncio con el nuevo camino (Retirada implícita). Pero esto puede no ser posible por la limitación del MRAI. Sin embargo podemos enviar una retirada explícita que *eliminará(flush)* la información fantasma generada por nosotros que ya no es válida.

El significado de los mensajes de retirada cambia respecto al que el BGP4 tradicional le daba. Antes, un mensaje de retirada significaba que un nodo no conocía ningún camino al destino referido. Ahora bajo la regla ghost-flushing, un mensaje de retirada significa que el camino de ASs antes anunciado ya no es válido. Un router puede tener caminos a un destino y enviar retiradas sobre ese destino.

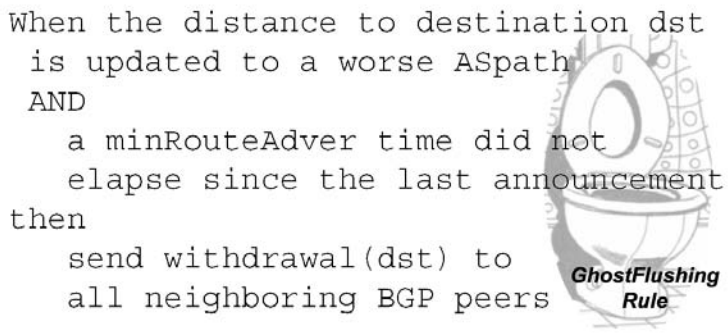

Figura 3.3: Regla ghost-flushing en su enunciado original en inglés.

Mostremos un ejemplo gráfico de como funciona. En *fig 3.4* podemos ver la misma situación que en *fig 3.1*, el clique de cinco nodos. Hasta t=2, todos los nodos hacen lo mismo que con BGP4 tradicional. Pero entonces, todos los nodos (Menos el nodo 1) cambian su camino para llegar a 0 por un más largo, pero no pueden enviar un nuevo anuncio, porque en t=1 ya enviaron uno. Aquí es donde entra la regla ghost-flushing y una retirada general de la ruta es enviada por todos los nodos(Porque información fantasma ha sido detectada). Como consecuencia, en t=3, todos los nodos han vaciado sus tablas de encaminamiento y dejan de hacer nada más. Solamente en 3 segundos, un resultado mucho mejor que en *fig 3.4*.

Analizando el ejemplo podemos distinguir dos fases:

Distribución de Información Fantasma aparece por la primera retirada implícita. Esta información activa la regla ghost-flushing en todos los nodos. En el ejemplo esto ocurre entre t=1 y t=2.

General Flushing(Eliminación general) por los mensajes de retirada producidos al detectar la información fantasma. Todos los nodos limpian la información fantasma en sus tablas. Esto ocurre en t=3.

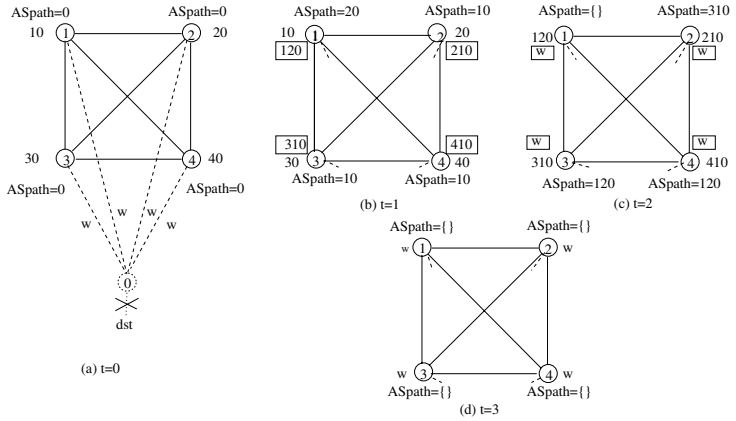

Figura 3.4: Paso a paso la regla ghost-flushing en acción.

3.2.2 Tiempo de convergencia

En [13] hay un análisis matemático profundo con demostraciones completas de los resultados que vamos a mostrar en esta sección. En la notación que usaremos debemos definir:

h latencia de las conexiones.

n número de nodos.

E número de arcos.

minRouteAdver intervalo mínimo de anuncio de rutas.

Mostraremos ahora todos los tiempos de convergencia relevantes para este trabajo

Situación Fail-Down

$$E_{down} = n * h \tag{3.1}$$

$$mensajes = \Theta(\frac{2hnE}{minRouteAdver}) \tag{3.2}$$

La idea sobre E_{down} es simple. Después de k turnos todos los caminos AS menores que o iguales a k han sido eliminados. Al principio los nodos más cercanos eliminarán sus rutas y esta eliminación (Flush) se irá propagando, eliminando caminos más largos en nodos más lejanos. En el peor caso, el nodo más lejano están a n saltos de distancia(h segundos cada salto), por lo que acaba en $n * h$ segundos.

Podemos ver que la complejidad es mucho mejor que con el BGP tradicional, con $minRouterAdver * n$.

Situación Fail-Over

Analíticamente no es posible probar que la regla ghost-flushing es siempre mejor que el BGP tradicional. Como se indicaba en [15], el tiempo de convergencia en este caso depende de cuanto cuesta hasta que la información fantasma desaparece del sistema y además, el nuevo camino se propague. Como veremos en otros capítulos, el caso mostrado en *3.2* mejora utilizando la regla ghost-flushing. Sin embargo hay casos en los que la regla se comporta peor que el BGP tradicional. Un ejemplo puede verse en *fig. 3.5*. La cadena de eventos que ahí ocurre es la siguiente:

- El Nodo X cambia el camino a través de K al camino por M. Lo anuncia a S.
- Entonces, X se da cuenta de que ese camino tampoco es bueno. Entonces ahora elige uno a través de Y que es más largo.
- El último evento dispara la regla ghost-flushing, enviando un mensaje de retirada a S.

- X envía la nueva ruta nueva a S después del temporizados MRAI.

Con la regla ghost-flushing este caso resulta en un peor comportamiento porque S siempre debería encaminar a través de X para alcanzar Dst. Pero durante MRAI segundos no tendrá esa ruta por el mensaje de retirada generado por la regla ghost-flushing en X. Podemos afirmar que esto ocurrirá *siempre y cuando desde un solo nodo, el camino AS se divida en más de dos caminos alternativos y al menos dos de ellos fallan a la vez, quedando un tercero más largo.*

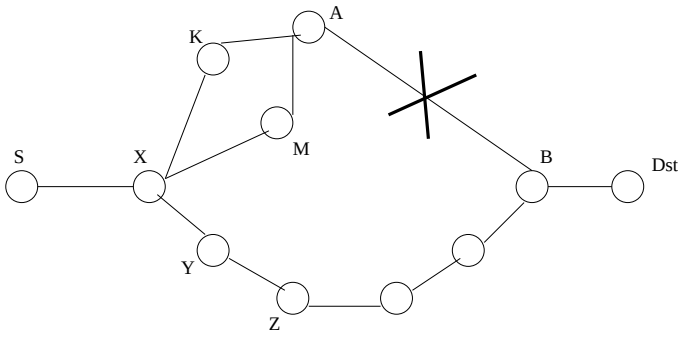

Figura 3.5: Un mal caso para la regla Ghost-Flushing.

Start-up y Shorter-Path

Para estos casos no existe un cambio a un camino más largo en las tablas de encaminamiento. Esto significa que la regla ghost-flushing no se activará, por lo tanto no hay diferencia alguna con el BGP tradicional.

3.3 Regla Ghost-Buster

3.3.1 Como Funciona

En el artículo [15] otra regla era definida para mejorar el comportamiento de BGP en los casos estudiados. Es la **regla Ghost-Buster**. Esta regla funciona suponiendo que la regla ghost-flushing está también siendo aplicada. La idea que origina esta regla es que, mientras las retiradas funcionan como limpieza de información fantasma, los mensajes de anuncio podrían ser retrasados aún más de lo obligatorio. De este modo la información fantasma es totalmente bloqueada. Éste es un enfoque todavía más agresivo para detener la información fantasma respondiendo a dos situaciones: En primer lugar, algunas implementaciones de BGP4 no incluyen la limitación de MRAI, de esta manera nos aseguramos de ello. La segunda situación es intentar modelar el escenario en el que se producen interacciones entre la regla ghost-flushing y algunas estrategias de control de oscilación de ruta[16], provocando retrasos en la retransmisión de rutas.

Delta puede tomar cualquier valor, sin embargo, como las simulaciones mostrarán más adelante, si es menor que el valor MRAI no se percibirá ningún efecto derivado de esta regla. De nuevo, una definición más compleja y una descripción matemática completa pueden ser encontradas en [15].

3.3.2 Tiempo de Convergencia

Situación Fail-Down

Siendo $K = \frac{delta+h}{h}$

$$E_{down} = hd\frac{K}{K-1} \tag{3.3}$$

La idea detrás de esta fórmula es que, de acuerdo con la regla ghost-buster, el camino AS fantasma solamente puede crecer una vez cada $delta + h$. La máxima dimensión de un camino AS es d, entonces la ecuación para t (Tiempo de convergencia) es:

Figura 3.6: Regla Ghost-Buster en su enunciado original en inglés.

$$d + \frac{t}{delta + h} = \frac{t}{h} \tag{3.4}$$

Por definición de K podemos reemplazar $delta + h = Kh$ y entonces:

$$d + \frac{t}{Kh} = \frac{t}{h} \tag{3.5}$$

y entonces:

$$t = E_{down} = hd\frac{K}{K-1} \tag{3.6}$$

Otras situaciones

Esta regla requiere que *todos los anuncios deben ser retrasados*, esto debe cumplirse sean o no sobre información fantasma. Las implicaciones incluyen un impacto negativo en otras situaciones como start-up, shorter path or fail-over. El interés de esta regla es doble, por un lado entender las propiedades de convergencia del BGP y ver las posibles interacciones de ghost-flushing con otras técnicas de control de oscilación de ruta.

3.4 Otras Soluciones

3.4.1 Reglas de Reset

Las reglas de reset [15] extienden a la regla ghost-buster aplicando el temporizador delta a todo tipo de actualización, sea anuncio o retirada. No tiene un interés aplicado al mundo real, así que no fue analizada.

3.4.2 Otros enfoques sobre el mismo problema

Reverse Poisoning (IGRP) Es una solución no escalable, primero limpia todas las rutas de los routers y empieza de nuevo. Es funcional en encaminado interno, pero definitivamente no para externo.

Aserciones de Consistencia sobre Rutas para poder identificar totalmente a la información fantasma, requerirían un tiempo de computación extra que podría dañar al rendimiento de los routers. También lleva a problemas en caso de partir las AS [13].

Detección Activa de Ciclos (Envio/recepción)[15] Reduce el tiempo de convergencia, pero como veremos en la fase de simulación, ghost-flushing funciona mejor.

Propuestas en [12] implican información extra en los mensajes de retirada. En este trabajo queremos evitar cualquier modificación en los mensajes del protocolo.

Capítulo 4

Entorno de Simulación BGP4

4.1 Simulador y casos Simulados

Tal y como se indica en la memoria principal utilizamos el simulador SSFNet 2.0. Este simulador utiliza un formato de definición de redes llamado DML. Para generar los ficheros DML se desarrollaron una serie de programas que leían formatos propios más sencillos.

Durante las simulaciones, se utilizaron varios escenarios en diferentes condiciones. Los escenarios (Descritos más ampliamente en la memoria completa) son: clique, clique con camino alternativo, caso malo para ghost-flushing, un caso extraído de un artículo y una serie de modelos extraídos de tablas de encaminado reales de Internet. Además estos escenarios se simularon en varias condiciones según si se usaban o no 2 funcionalidades propias de BGP4: split-horizon y aplicación de jitter a los temporizadores. Para más detalles acudir a la memoria principal.

4.2 Modificaciones a la implementación para la regla Ghost-Flushing

Para implementar esta regla dentro del simulador tuvimos que analizar el proceso de actualización (Desde la llegada de una actualización hasta el envío de una nueva). Llegamos a la conclusión de que para hacer al protocolo cumplir la regla ghost-flushing tenías que implementar dos funciones: Identificar la información fantasma y enviar nuevos mensajes de retirada cuando fuesen necesarios.

Para **identificar la información fantasma** decidimos que el mejor momento en el código del BGP era dentro de la segunda fase del proceso de decisión (decision_process_2()). En ese momento tenemos acceso al LocRIB (Ruta utilizada hasta ahora) y la nueva ruta que va a ser elegida (Y que puede ser información fantasma). Si descubrimos que la antigua ruta va a ser sustituida por una ruta peor, hemos creado información fantasma anteriormente. Para mantener un control de la información también usaremos una estructura de datos almacenando los nlris sobre los que hemos recibido información fantasma. Para cada NLRI almacenaremos la longitud del último camino AS que hemos puesto en la LocRIB (Puede haber muchos reemplazos en poco tiempo) y la longitud que estaba en el LocRIB antes de cualquier modificación.

El lugar para insertar el **envío de nuevos mensajes de retirada** era fácil de encontrar. Se decidió que el mejor lugar era durante la tercera fase del proceso de decisión (decision_process_3()). In ese momento el protocolo decide como enviar las rutas y las retiradas. Si la nueva ruta sobre un NLRI en la LocRINB es más larga que la que teníamos antes, enviaremos un mensaje de retirada sobre esa NLRI a todos los nodos.

4.2.1 Activando la regla Ghost-Flushing desde el archivo DML

El sistema de lectura del archivo de configuración también fue modificado. Dos entornos de configuración fueron modificados, el global y el local. Las aserciones locals son utilizadas por cada uno de los nodos. Las globales afectan a todos y cada uno de los nodos definidos. Éste último es especialmente útil cuando se trabaja con ficheros DML pregenerados. En esos casos modificar todos los nodos puede ser costoso. Las aserciones definidas son:

bgpoptions environment Sobreescribe configuración local.

- forceghostflushing true/false, fuerza el uso de esta regla.
- ghostdebug true/false, activa o desactiva los mensajes de debugging asociados con esta regla.

BGPSession environment

- ghostflushing true/false, fija el uso de esta regla.

4.2.2 Esquema de modificación Modification

En *fig. 4.1* podemos encontrar un esquema simplificado de como SSFNet trata los mensajes de actualización con las modificaciones en negrita. (El completo está en la versión completa de la memoria).

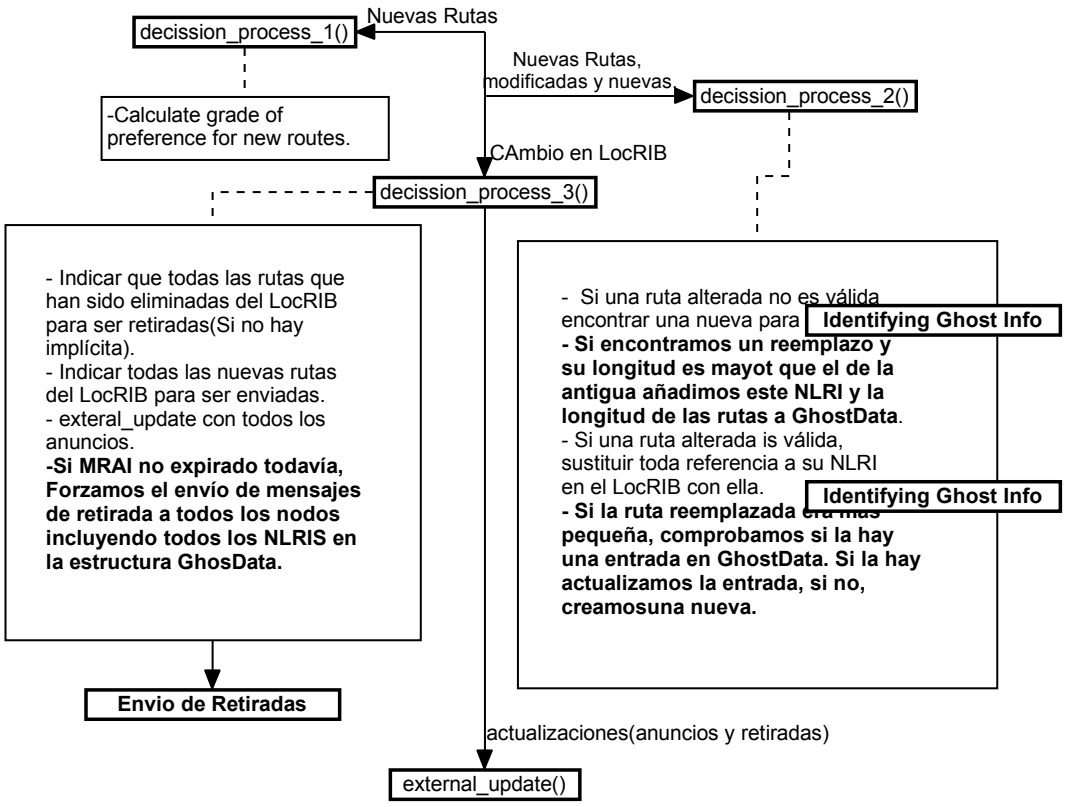
decission_process_1()
Nuevas Rutas
Nuevas Rutas, modificadas y nuevas
decission_process_2()
-Calculate grade of preference for new routes.
CAmbio en LocRIB
decission_process_3()
- Indicar que todas las rutas que han sido eliminadas del LocRIB para ser retiradas(Si no hay implícita).
- Indicar todas las nuevas rutas del LocRIB para ser enviadas.
- exteral_update con todos los anuncios.
-Si MRAI no expirado todavía, Forzamos el envío de mensajes de retirada a todos los nodos incluyendo todos los NLRIS en la estructura GhosData.
- Si una ruta alterada no es válida encontrar una nueva para
Identifying Ghost Info
- Si encontramos un reemplazo y su longitud es mayot que el de la antigua añadimos este NLRI y la longitud de las rutas a GhostData.
- Si una ruta alterada is válida, sustituir toda referencia a su NLRI en el LocRIB con ella.
Identifying Ghost Info
- Si la ruta reemplazada era mas pequeña, comprobamos si la hay una entrada en GhostData. Si la hay actualizamos la entrada, si no, creamosuna nueva.
Envio de Retiradas
actualizaciones(anuncios y retiradas)
external_update()

Figura 4.1: Tratamiento de actualizaciones con Ghost-Flushing en el esquema BGP de SSFNet.

4.3 Modificaciones al la Implementación regla Ghost-Buster

4.3.1 Activando la regla Ghost-Buster desde el archivo DML

El sistema de lectura del archivo de configuración también fue modificado. Dos entornos de configuración fueron modificados, el global y el local. Las aserciones locales son utilizadas por cada uno de los nodos. Las globales afectan a todos y cada uno de los nodos definidos. Éste último es especialmente útil cuando se trabaja con ficheros DML pregenerados. En esos casos modificar todos los nodos puede ser costoso. Las aserciones definidas son:

bgpoptions environment Sobrescribe la configuración local.

- forceghostbuster true/false, fuerza el uso de esta regla.

- ghostdebug true/false, activa o desactiva los mensajes de debugging asociados con esta regla.

BGPSession environment

- ghostbuster true/false, fija el uso de esta regla.

4.3.2 La idea detrás de la modificación

En este caso, modificar el protocolo para que cumpla con la regla ghost-buster no fue tan fácil como con la regla ghost-flushing. Siguiendo la estrategia de la anterior modificación, intentamos identificar la tareas principales necesarias para esta regla. Definimos cuatro tareas principales: Creación e inicialización del temporizador Delta asociado con cada ruta recibida por el protocolo, modificación del procedimiento de envío para que respetase la limitación delta, tratamiento del evento de expiración de un temporizador delta. Para todo ello fue necesaria la implementación de los temporizadores, mensajes y clases necesarias para hacer esto funcionar.

Para la **creación e inicialización del temporizador delta** asociado con cada ruta, se pensó que el mejor lugar para implementarlo era dentro del código de la clase AdjRIBIn. Modificamos el código de esta clase para que cada vez que una ruta era añadida al AdjRIBIn, un temporizador delta era creado y asociado a la ruta. Funciona del mismo modo cada vez que una ruta era eliminada del AdjRIBIn, el temporizador asociado es eliminado.

Para **el método de envío**. Alteramos el método try_send_update para que el temporizador delta para una ruta que cuyo temporizador no está expirado, no sea enviado o puesto en la lista de espera.

Tratando la expiración del temporizador delta, para esto se creo un nuevo evento interno dentro de BGP4. La idea es bastante simple, añadimos una entrada dentro de método handle_event que trata cada ruta cuyo temporizador expira. Si la ruta está en Loc_RIB se intenta enviar de acuerdo con las limitaciones clásicas de BGP4 (MRAI, filtros, etc) como si el temporizador delta nunca hubiera existido.

Mensajes, Temporizadores y cintas de video. Para que todo esto funcione necesitamos varias clases nuevas. Necesitamos un nuevo tipo de temporizador que disparase el evento apropiado dentro de la máquina de estados finitos:**DeltaTimer**. Es creado y asociado a una ruta cuando esta llega. Cuando expira produce un mensaje que también tuvo que ser definido (Otra clase): **DeltaTimeoutMessage**. El contenido de este mensaje incluye un referencia a la ruta que ha disparado el evento, de ese modo handle_event sabe lo que tiene que hacer. Para mantener en una estructura de datos todos los temporizadores junto con la referencias cruzadas entre ellos y las rutas asociadas, creamos la clase **BusterRoute**, que ayuda a manejar la creación y destrucción de los temporizadores. Para indexar todos las instancias de BusterRoute en la clase que representa un vecino al que bgp está conectado usamos la clase **ListBusterRoute**. Consiste en una tabla hash indexada por los NLRIs de las rutas y tiene varios métodos para hacernos la vida un poco más fácil.

4.3.3 Como funciona todo este lío

Vamos a intentar mostrar la vida de una ruta desde que es recibida hasta que es es enviada a otros vecinos: El nuevo "ciclo de vida" de una ruta. Considerando la complejidad de la modificación solamente incluimos lo que es diferente de la regla ghost-flushing.

Ruta llega y el protocolo intentan enviarla pero el temporizador delta no ha expirado

Este esquema puede ser encontrado en *fig. 4.2*

Temporizador Delta Expira

Suponemos que la ruta está en el LocRIB (Por lo tanto queremos enviarla). Este esquema puede ser encontrado en *fig. 4.3*

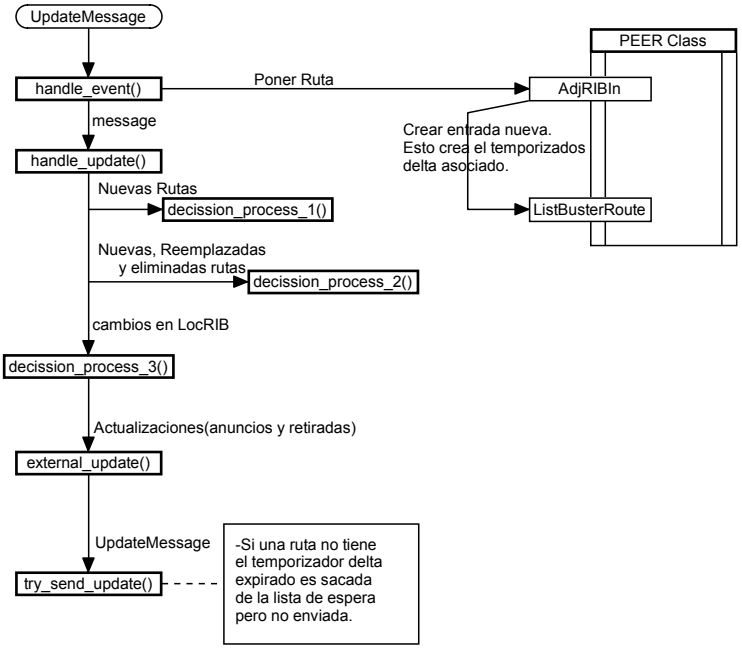

Figura 4.2: Un ruta llega y el protocolo intenta enviarla, aunque el temporizador delta no ha expirado.

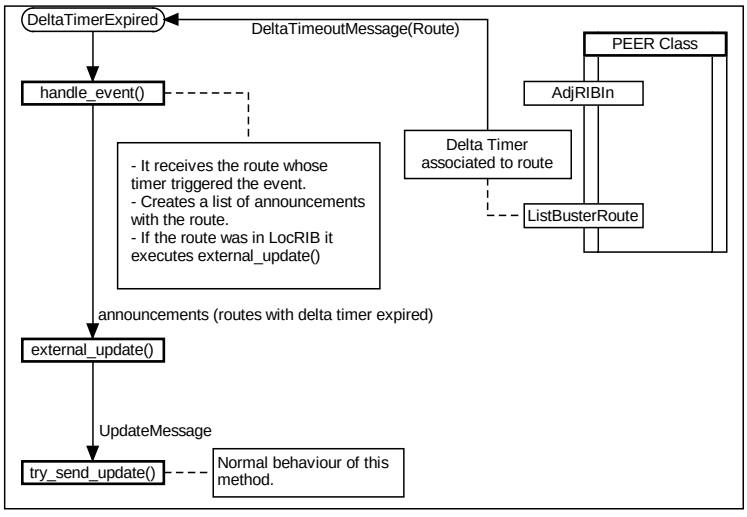

Figura 4.3: Temporizador Delta Timer Expira.

Capítulo 5

Simulación de Casos de BGP4

Este capítulo es un resumen de su homólogo en la memoria principal. Para más detalles sobre el método de simulación, escenarios y resultados más detallados acudir a dicha memoria.

5.1 Comparando todas las técnicas

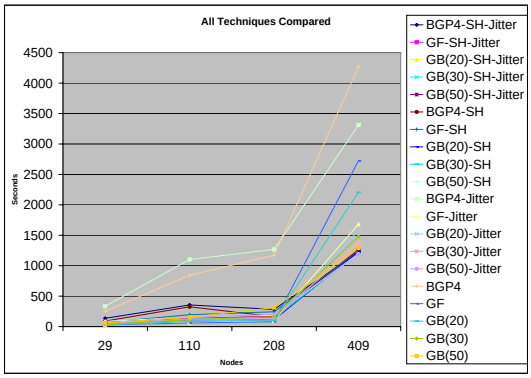

Figura 5.1: Multi E_{down}.

En la memoria completa comparamos todas las técnicas junto con el protocolo BGP tradicional en muy diferentes situaciones. Para más detalle, acudir a la memoria principal. Durante los experimentos repetimos las simulaciones cambiando algunas funcionalidades dentro de BGP4: **aplicación de jitter a los temporizadores y split-horizon**.

En esta sección compararemos todas las técnicas con estas funcionalidades activadas y desactivadas. La idea es ver cual es la mejor combinación. En *fig.5.1* podemos ver el resultado de todas las configuraciones. Para cualquier versión, el mejor rendimiento aparece con split-horizon y jitter activado. No ha aparecido ningún comportamiento peor al combinar las reglas. Es seguro entonces utilizarlas todas juntas.

5.2 Combinando nodos con y sin ghost-flushing

Un punto interesante en este trabajo de investigación era ver que ocurriría si combinamos routers BGP que usan ghost-flushing con nodos que no lo utilizan. Teniendo en cuenta que los routers y redes en Internet pertenecen a diferentes instituciones y cada una puede utilizar diferentes marcas de routers, es difícil que todos pudieran aplicar la regla ghost-flushing a la vez. Este experimento consiste en coger el escenario más efectivo para la regla ghost-flushing, el clique, y empezar a desactivar la regla en los nodos.

Elegimos cliques entre 4 y 15 nodos. El experimento es el mismo que antes. Ahora, comienza con todos los nodos usando ghost-flushing is es repetido reduciendo cada vez el número de nodos usando la regla.

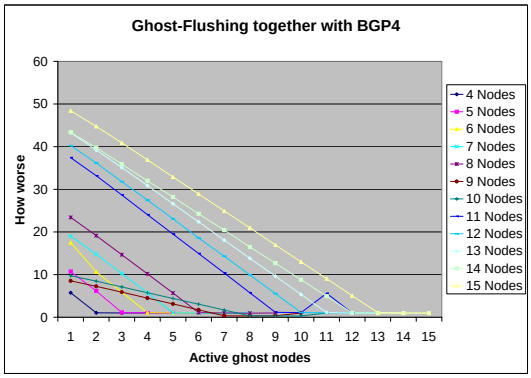

Figura 5.2: Mezclando BGP4 tradicional con ghost-flushing. Más es peor.

La *fig. 5.2* indica el resultado de nuestro experimento. El contenido de la figura es confuso, así que vamos a explicarlo. Cada línea representa la evolución de cada clique, empezado (izquierda) con ningún nodo usando ghost-flushing e incrementando el número de nodos después. Como es natural, cada serie termina con el caso en que todos los nodos usan ghost-flushing. El valor "How worse" significa cuantas veces mayor es E_{down} con ese número de nodos usando ghost-flushing comparado con el mejor E_{down} posible (todos con ghost-flushing) dado en ese clique.

Como podemos ver en la figura hay una reducción lineal de E_{down} al aumentar el número de nodos usando ghost-flushing. Alcanzando el punto donde solo el 20% de los nodos no usan ghost-flushing y el resultado se acerca al óptimo. Esto es una buena noticia de cara a la implementación de este protocolo en Internet. Podemos ver que hay una mejora en el rendimiento incluso si todos los nodos no utilizan esta regla.

5.3 Conclusiones

Estas conclusiones son un resumen de los resultados obtenidos en el capítulo completo de simulación en la memoria principal.

5.3.1 Ghost-Flushing

Como era esperado el comportamiento de la regla ghost-flushing ha sido bastante bueno. Comparado con el BGP4 tradicional, ha mejorado en todos los aspectos (Tiempo de Convergencia y número de mensajes) sin importar el escenario. Solamente en algunas situaciones, aplicando split-horizon y jitter, el BGP4 se acercaba al rendimiento de la regla ghost-flushing. Sin embargo, esto solo se daba en escenarios de reducidas dimensiones, en cuanto la topología de red crecía, la regla ghost-flusher volvía a distanciarse del BGP4 tradicional.

En las **situación de Set-Up** no hay diferencia con el BGP4 tradicional. En las situaciones de **fail-over** y **fail-down** hay una gran diferencia. La complejidad de la convergencia cambia de lineal a constante (en el caso más simple), dependiendo esta de la conectividad. Parece como si cuanto más conectados estén los nodos mejor funciona la regla ghost-flushing (Y pero BGP4 tradicional).

Además, los **experimentos mezclando ghost-flushing** con BGP4 tradicional son alentadores. Incluso si no todos los nodos usan la regla, el rendimiento del sistema mejora. Hay una relación lineal entre el número de nodos que usan la regla y la mejora en el sistema. Es un gran resultado de cara al despliegue de este protocolo en Internet.

Sobre la implementación. Resulta sencilla, no implica complicados algoritmos que usen extra cpu o cantidades de memoria.

Revisando todos estos resultados y conclusiones, podemos decir que **merece la pena implementar esta regla** en GNU Zebra. Los resultados son buenos y los inconvenientes suficientemente pequeños como para hacerlo.

5.3.2 Ghost-Buster

Los resultados sobre la regla ghost-buster son bastante sorprendentes. Inicialmente, cuando ejecutamos las primeras simulaciones, encontramos los resultados esperados, un rendimiento entre el BGP4 tradicional y la regla ghost-flushing (O peor). Sin embargo, cuando comenzamos a simular modelos más grandes y aplicar la regla split-horizon y jitter en los temporizadores **el rendimiento de la regla ghost-buster sufre un salto espectacular.**

Empecemos con la situación de **Fail down**. Hay casos donde la regla ghost-buster con el mayor valor para el temporizador delta (50 segundos) sobrepasa en rendimiento al resto de las reglas. El modelo donde este efecto se da más es en la topología "multi". Cuando los modelos de mayores dimensions son probados, encontramos casos en los que la regla ghost-buster se comporta mejor en tiempo y mensajes que ghost-flushing. Parece que el enfoque conservador a la hora de mandar anuncios es mejor en situaciones de alta interconectividad donde hay muchos caminos posibles y muy largos.

Lo mismo ocurre en el escenario de **set-up**. De nuevo, en un escenario extremadamente grande, la regla ghost-buster se comporta mejor tanto en número de mensajes como tiempo.

Mirando a los resultados podemos extraer una serie de **conclusiones generales**. Parece the dependiendo del punto de vista, la regla ghost-buster se comporta mejor o peor. Si miramos al tiempo que cuesta que toda la red se estabilice, ghost-buster se comporta bastante bien. Sin embargo, si miramos al aspecto más local, a pequeñas partes de red (Modelos menores), la regla ghost-buster se comporta peor.

No simulamos ningún caso de como se comportaban los modelos de 400 nodos si reducimos el número de nodos con ghost-buster en a red. Es simplemente un tema de tiempo. Simular una vez un escenario de 400 nodos cuesta casi una hora, necesitaríamos ejecutar 400 simulaciones más de una vez para conseguir resultados estables y eso cuesta mucho tiempo.

Algo importante sobre esta regla es que **necesita mucha CPU y memoria**. Cualquier camino recibido necesita un temporizador que ocupa memoria y produce un evento cuando expira (Aunque no sea necesario). Esto puede tener un impacto negativo en el rendimiento del router.

Mirando a todas estas conclusiones vemos ventajas y desventajas. Vemos un **mal rendimiento en escenarios medianos y pequeños escenarios** (menos de 200 nodos) para la situación de set-up y fail-down. Esto junto con la complicada y pesada implementación, nos lleva a tomar la decisión de no implementar esta regla.

Capítulo 6

Aplicando la regla Ghost-Flushing a GNU Zebra

Después del análisis de la simulación de las diferentes modificaciones propuestas para BGP4 obtuvimos un claro ganador, la regla ghost-flushing. Entonces era momentos de hacer una implementación sobre un software de encaminado real. Necesitábamos un software gratuito y de código abierto, mirando al mercado encontramos: GNU Zebra, Quagga, GateD y OpenBSD's bgp. Por diferentes motivos (Memoria principal) elegimos GNU Zebra como nuestro software de encaminado.

6.1 Zebra como Software de Encaminado

GNU Zebra es software libre y de código abierto que implemente una serie de protocols de encaminado. Incluye los más utilizado como BGP4, RIP(v1 and v2) y OSPFv2. En el caso de BGP4 sigue las especificaciones en el RFC-1771[28]. Una de sus características más atractivas es su enfoque modular. Cada protocolo de encaminado es implementado en un demonio diferente, siendo el demonio Zebra su unión con la tabla de encaminado del kernel. Esto hizo que modificar el demonio bgpd no implicase alterar nada más en el software de encaminado que no tuviera que ver con el propio protocolo, no interfiriendo para nada con otras funciones o protocolos.

6.1.1 Funcionalidades

Su características más importante es la modularidad, cada protocolo es implementado en un demonio diferente, esto implica parte de las varias ventajas de este paquete:

- **Fácil de modificar.**
- **Reparto de carga asimétrico**.
- **Fiabilidad**.
- **Software gratuito y abierto**. item **Totalmente adaptable**.

Otros Detalles:

- Implementa los RFCs: RFC-1771[28], Route Reflector y Confederaciones [30, 25], y control de oscilación de ruta [16].
- **IPv6 aware**.
- Soporta: FreeBSD, NetBSD, OpenBSD, GNU/Linux sobre IPv4 y IPv6. Aunque elegimos usarlo FreeBSD por su buen rendimiento en sistemas de redes.
- El sistema de configuración es un clon de la notación de los routers Cisco.
- Como desventajas debemos decir que aunque es un buen software no puede competir con un router empotrado con hardware dedicado y preparado para ejecutar el software de encaminado.

6.2 Arquitectura de GNU Zebra

Este capítulo intenta dar una idea de como está estructurado Zebra, para detalles sobre la arquitectura de bgpd, recurrir a la memoria principal. Estos contenidos están basado en [14] y nuestra experiencia con GNU Zebra.

6.2.1 Módulos, Demonios y Sockets

Como hemos indicado antes, **Zebra es un software extremadamente modularizado**. Cada tarea es ejecutado por un demonio diferente. Cada protocolo de encaminado es implementado por un demonio diferente. Además, la función de intercambiar información con las tablas del kernel y entre los protocolos es ejecutada por otro demonio, el demonio Zebra. Esto ayuda a dividir el código según la función que desempeñe. La comunicación entre todos los demonios se realiza sobre sockets y usando el protocolo Zebra.

Veamos un pequeño ejemplo. En emphfig. 6.1 tenemos al Zebra trabajando con 3 demonios, uno para BGP otro para OSPF y otro para RIP. bgpd aprende la ruta A a través de sus sesión BGP. Después de realizar sus funciones y elegirla como ruta, envía dicha ruta al demonio Zebra usando el protocolo Zebra. Entonces este demonio actualiza las tablas de encaminado del kernel y envía la ruta a los demás demonios.

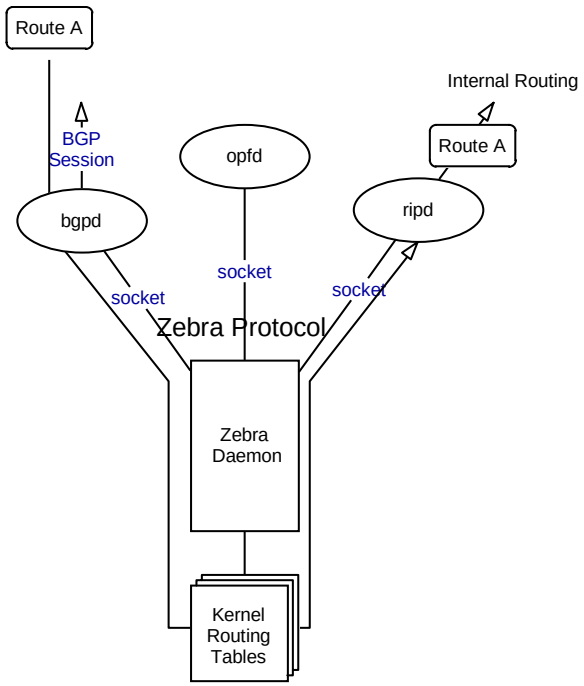

Figura 6.1: Estructura modular de GNU Zebra. Todo acaba pasando por el demonio Zebra.

Algo muy interesante sobre este enfoque es que no hay necesidad de que todos los demonios estén en la misma máquina, repartiendo así la carga. Una máquina podría hacer de router BGP4 pero no de encaminador para la red. Otra máquina tendría el demonio Zebra, por lo tanto sus tablas estarían actualizadas. A través de los sockets ambos demonios se comunicarían a través de la red. Dividimos la carga de las funciones de BGP y de encaminado a diferentes máquinas (Que pueden tener diferentes configuraciones de software-hardware).

6.3 Tests

6.3.1 Tests y resultados

Durante esta fase teníamos dos objetivos.

- Comprobar si la implementación de ghost-flushing realizada sobre Zebra se comportaba como debía (Corrección).
- Realizar pequeñas pruebas de rendimiento para ver si la mejora teórica aparecía en la realidad.

Corrección de la Implementación

Para este propósito utilizamos la topología de clique de 5 nodos del mismo modo que la usamos para validar el simulador (Memoria completa), es decir establecer un nodo y parar uno de ellos para ver que ocurría. Activamos la función de debugging del protocolo. Después de revisar los archivos de log, comprobamos que la implementación funcionaba correctamente.

Pruebas de Rendimiento

En esta situación probamos dos escenarios: clique de 5 y 7 nodos. Repetimos los tests de la sección anterior midiendo cuanto tiempo costaba que la red se estabilizase.

- 5 Nodos:
  - E_{down} con ghost-flushing: 1-3 segundos.
  - E_{down} sin ghost-flushing: 50-60 segundos.
- 7 Nodes:
  - E_{down} con ghost-flushing: 2-4 segundos.
  - E_{down} sin ghost-flushing: 110-120 segundos.

Mirando a los archivos de log vemos el comportamiento de convergencia descrito en el apartado teórico.

6.4 Conclusiones

6.4.1 Conclusiones Generales

Debemos resaltar que pudimos reproducir en un escenario real el comportamiento anómalo en la convergencia de BGP4. Esto demuestra que es un problema que no solo existe sobre el papel y que se puede dar con un software de encaminado real.

Sobre la capacidad de **GNU Zebra** como software de encaminado, nuestra experiencia (Desde dentro y desde fuera) muestran este software como una opción fiable y barata para configurar un router sobre un hardware convencional. Esta opinión está también soportada por artículos en la comunidad de encaminado [34] y los numerosos usuarios que lo utilizan(Zebra mailing list [4]). Además la posibilidad de modificar el código del router a nuestro gusto y según nuestras necesidades hace de GNU Zebra una buena opción.

Sobre la regla **ghost-flushing**, muestra las mismas propiedades encontradas durante la simulación. Los resultados muestran la mejora supuesta a la regla.

Capítulo 7

Conclusiones Y Discusiones

7.1 Calidad de los resultados de Simulación

¿Cuál es la **calidad de los resultados** obtenidos de las simulaciones realizadas sobre SSFNet? Mirando las características del simulador, intenta emular todas las capas, desde el protocolo de encaminado y los diferentes niveles de la pila de red. Suponiendo que todo está bien implementado y que el motor de simulado es correcto, podemos confiar el los resultados.

Al principio de este trabajo intentamos realizar la única prueba de validación que nos fue posible (Memoria completa). La prueba consistía en simular un caso en el que conocíamos todo lo que tenía que ocurrir y comparar con los resultados del simulador. El resultado fue positivo así que no descartamos este simulador como ocurrió con ns-2.

Otras fuentes de confianza en el simulador pueden venir de su reputación en la comunidad de encaminado. Parece ser que, al menos la parte exterior del BGP4, funciona correctamente.

La única nota añadir es que por problema derivados de la implementación de la máquina virtual de Java, no puede simular modelos de más de 500 nodos.

7.2 La regla Ghost-Flushing

Como las simulaciones y los tests sobre Zebra muestran, es bastante efectiva. Tine varios puntos positivos:

- **Bajo Consumo de CPU**, no hay algoritmos complejos en esta regla, esto tiene un impacto casi nulo en la CPU.
- **Bajo uso de memoria**. De hecho, no hay ningún uso extra. Para la regla ghost-buster esto era un problema, porque necesitaba crear temporizadores (con su memoria asociada) para cada ruta recibida.
- **No información extra** en los mensajes que pueda llevar a incompatibilidades con el BGP4 convencional.
- **Buen rendimiento** como se ve en el capítulo de simulación y test de GNU Zebra.

En total, podemos decir que la regla ghost-flushing es bastante efectiva y una fácil modificación de BPG4. Desde nuestra humilde posición, recomendaríamos la implementación de esta regla en los actuales softwares de encaminado al igual que nosotros hicimos con GNU Zebra.

7.3 Software de Encaminado GNU Zebra

¿Es GNU Zebra una opción real como software de encaminado? Mirando a nuestra propia experiencia, la opinión de algunos "expertos"[34] y la comunidad de usuarios que posee, podríamos decir que sí, siempre y cuando necesites un router barato. Implementa todas las RFCs relacionadas con BGP4 y los protocolos principales de encaminado. Su estructura modular lo hace **fiable** y fácil de diagnosticar en caso de fallo. La posibilidad de poder modificar su código para ajustarlo a nuestras necesidades le da un aspecto atractivo. El único problema es que por muy bueno que sea el software, **nunca superará a**

un router empotrado: Es decir una colección de hardware y software específicamente diseñada para encaminar información. Sin embargo, GNU Zebra siempre será más barato.

Sobre la implementación de ghost flushing, fue fácil de hacer (Aunque no de saber como hacerla) y parece funcionar. Nuestros tests además, muestran un buen rendimiento.

Un comentario adicional sobre GNU Zebra gira alrededor de la documentación existente. Hay mucha documentación sobre su estructura general, sin embargo no existe nada disponible sobre como han sido implementadas las diferentes funcionalidades de los protocolos. Teniendo en cuenta que el paquete está desarrollado en C, analizar el código para averiguar como funcionar no es la operación más sencilla del mundo. GNU Zebra comparte un problema que muchos paquetes de código abierto sufren, suelen estar bien comentados, pero aunque el código se publique la documentación de desarrollo asociada se mantiene en privado. Esto limita en muchos casos la posible colaboración de muchos desarrolladores ante el problema de emplear más tiempo analizando el código hecho que desarrollando (Como fue el caso en este proyecto). Por ello,la documentación de este proyecto (Memoria completa) intenta ser un pequeño mapa de como algunas de las funciones de BGP4 están implementadas en bgpd de GNU Zebra.

Capítulo 8
Trabajo Futuro

8.1 Interacción de Ghost-Flushing con otras reglas

BGP4 tiene muchas funcionalidades que pueden interactuar con la regla ghost-flushing. Hemos tenido en cuenta algunas de ellas (Split-horizon, jitter) pero no todas.

En primer lugar, una linea de investigación podría ser el **análisis teórico de la interacción entre ghost-flushing y split-horizon**. Quizás fusionarlas en una sola solución podría llevar a una regla todavía más efectiva.

Otras funcionalidades de BGP4 que tienen que ver con el caso de la convergencia, como las políticas de filtrado [12], podrían ser analizadas en interacción con ghost-flushing. Aunque esto se podría hacer durante la fase de despliegue, ya que ocurrirán en la realidad de Internet.

8.2 Probando Zebra hasta su Límites

Para probar totalmente la implementación de ghost-flushing y las capacidades de Zebra, podríamos necesitar más equipos para crear mayores y más complejas redes.

Sin embargo, cuando este proyecto estaba casi terminado, conocimos de la existencia de una aplicación de simulación de entornos de red para Linux/Unix desarrollado por la Universidad Politécnica de Madrid. Este software es conocido como VNUML (Virtual Network User Mode Linux)[17]. Es un sistema para simular en un solo equipo entornos de red totalmente independientes, pudiendo hacer creer a varias instancias de una aplicación que están en diferentes máquinas.

Esto permitiría simular entornos con decenas de nodos con GNU Zebra, llevando a resultados más completos sobre carga y efectos en redes mayores.

Part III

Appendix

Appendix A
BGP with Ghost Flushing Pseudocode

```
Upon receiving message (type,PeerASpath,dst) from peer p in router r in AS_r
1   If (type == Withdrawal)
2       ASpath_dst_p = {}
3   If (type == Announcement)
4       If AS_r ∈ PeerASpath_dst {The loop detection/prevention mechanism }
5           ASpath_dst_p = {}
6               Else
7           ASpath_dst_p = PeerASpath {The ASpath to dst associated with peer p}
8   NewASpath_dst = Compute the Preferred ASpath_p from
                the announcements associated with all the peers
9   If (NewASpath_dst ≠ ASpath_dst)
10      ASpath_dst = NewASpath_dst
11      If (NewASpath_dst == {})
12          Send message (withdrawal,{},dst) to each peer
13          LastAnnouncedASpath_dst = {}
14          NextHop_dst = NULL {NextHop_dst to be used for routing packets to dst}
15      Else
16          NextHop_dst = p*,
            { where p* is the peer through which NewASpath_dst was announced }
16a         If (NewASpath_dst less preferred than LastAnnouncedASpath_dst)
            {An empty path ({}) is considered longer than any other path}
16b             If (currentTimeStamp − LastAnnouncedTime_dst < minRouteAdver)
16c                 Send message (withdrawal,{},dst) to each peer
16d                 LastAnnouncedASpath_dst = {}
17          If (currentTimeStamp − LastAnnouncedTime_dst ≥ minRouteAdver)
18              SendAnnouncement(dst)
19          Else
20              SendAnnouncement(dst)  at time LastAnnouncedTime_dst+minRouteAdver

21  SendAnnouncemnt(dst)
22  If (LastAnnouncedASpath_dst ≠ ASpath_dst)
23      send message (announcement,ASpath_dst,dst) to each peer
24      LastAnnouncedASpath_dst = ASpath_dst
25      LastAnnouncedTime_dst = currentTimeStamp
```

Figure A.1: Pseudocode implementation of Ghost-Flushing over BGP.

Appendix B

Zebra bgpd node configuration file example

```
# Router ID and IP
hostname bgpd
password lacasito
router bgp 1
bgp router-id 192.168.0.1
bgp ghost-flushing
bgp enforce-first-as
# Network Prefixes
network 130.240.0.0/16
# neighbor Routers
neighbor 192.168.0.2 remote-as 2
neighbor 192.168.0.3 remote-as 3
neighbor 192.168.0.4 remote-as 4
neighbor 192.168.0.5 remote-as 5
# Debug Info
#debug bgp
#debug bgp fsm
#debug bgp filters
#debug bgp keepalives
#debug bgp events
debug bgp updates
debug bgp ghost-flushing
log file bgpd.log.1
```

Figure B.1: Example configuration file for a zebra node. Its IP(ID) is 192.168.0.2. It has 5 neighbors. It uses Ghost-Flushing and it shows the debugging messages about it

Appendix C

DML 5 nodes clique scenario

```
_schema [ _find .schemas.Net ]
Net [
randomstream [
generator MersenneTwister
stream seed-lacasito
reproducibility_level timeline
]
frequency 1000000000
bgpoptions  [
        split_horizon               false
        jitter_mrai                 false
    show_id_data        true  # show AS number and prefix for each BGP router
    show_rcv_update     true  # show when Update msgs are rcvd
    show_snd_update     true  # show when Update msgs are sent
   ghostdebug false
        #
        proc_delay_model uniform_random
        max_proc_time           0.25
        min_proc_time           0.0
]
Net [
id 1
AS_status boundary
router [
id 1
graph [
ProtocolSession [
name bgp  use  SSF.OS.BGP4.BGPSession
autoconfig false
connretry_time 120  min_as_orig_time 15
ghostflushing true
reflector false
neighbor [ as 2 address 1(1) use_return_address 1(2)
_extends .basic_ebgp_neighbor ]
neighbor [ as 3 address 1(1) use_return_address 1(3)
_extends .basic_ebgp_neighbor ]
neighbor [ as 4 address 1(1) use_return_address 1(4)
_extends .basic_ebgp_neighbor ]
neighbor [ as 5 address 1(1) use_return_address 1(5)
_extends .basic_ebgp_neighbor ]
]
ProtocolSession [ name socket use SSF.OS.Socket.socketMaster ]
ProtocolSession [ name tcp    use SSF.OS.TCP.tcpSessionMaster ]
```

```
ProtocolSession [ name ip     use SSF.OS.IP ]
]
interface [ id 0  virtual true ]
interface [ id 2 ]
interface [ id 3 ]
interface [ id 4 ]
interface [ id 5 ]
]
]
Net [
id 2
AS_status boundary
router [
id 1
graph [
ProtocolSession [
name bgp  use  SSF.OS.BGP4.BGPSession
autoconfig false
connretry_time 120  min_as_orig_time 15
ghostflushing true
reflector false
neighbor [ as 1 address 1(2) use_return_address 1(1)
_extends .basic_ebgp_neighbor ]
neighbor [ as 3 address 1(2) use_return_address 1(3)
_extends .basic_ebgp_neighbor ]
neighbor [ as 4 address 1(2) use_return_address 1(4)
_extends .basic_ebgp_neighbor ]
neighbor [ as 5 address 1(2) use_return_address 1(5)
_extends .basic_ebgp_neighbor ]
]
ProtocolSession [ name socket use SSF.OS.Socket.socketMaster ]
ProtocolSession [ name tcp    use SSF.OS.TCP.tcpSessionMaster ]
ProtocolSession [ name ip     use SSF.OS.IP ]
]
interface [ id 0  virtual true ]
interface [ id 1 ]
interface [ id 3 ]
interface [ id 4 ]
interface [ id 5 ]
]
]
Net [
id 3
AS_status boundary
router [
id 1
graph [
ProtocolSession [
name bgp  use  SSF.OS.BGP4.BGPSession
autoconfig false
connretry_time 120  min_as_orig_time 15
ghostflushing true
reflector false
neighbor [ as 1 address 1(3) use_return_address 1(1)
_extends .basic_ebgp_neighbor ]
neighbor [ as 2 address 1(3) use_return_address 1(2)
```

```
_extends .basic_ebgp_neighbor ]
neighbor [ as 4 address 1(3) use_return_address 1(4)
_extends .basic_ebgp_neighbor ]
neighbor [ as 5 address 1(3) use_return_address 1(5)
_extends .basic_ebgp_neighbor ]
]
ProtocolSession [ name socket use SSF.OS.Socket.socketMaster ]
ProtocolSession [ name tcp    use SSF.OS.TCP.tcpSessionMaster ]
ProtocolSession [ name ip     use SSF.OS.IP ]
]
interface [ id 0  virtual true ]
interface [ id 1 ]
interface [ id 2 ]
interface [ id 4 ]
interface [ id 5 ]
]
]
Net [
id 4
AS_status boundary
router [
id 1
graph [
ProtocolSession [
name bgp  use  SSF.OS.BGP4.BGPSession
autoconfig false
connretry_time 120  min_as_orig_time 15
ghostflushing true
reflector false
neighbor [ as 1 address 1(4) use_return_address 1(1)
_extends .basic_ebgp_neighbor ]
neighbor [ as 2 address 1(4) use_return_address 1(2)
_extends .basic_ebgp_neighbor ]
neighbor [ as 3 address 1(4) use_return_address 1(3)
_extends .basic_ebgp_neighbor ]
neighbor [ as 5 address 1(4) use_return_address 1(5)
_extends .basic_ebgp_neighbor ]
]
ProtocolSession [ name socket use SSF.OS.Socket.socketMaster ]
ProtocolSession [ name tcp    use SSF.OS.TCP.tcpSessionMaster ]
ProtocolSession [ name ip     use SSF.OS.IP ]
]
interface [ id 0  virtual true ]
interface [ id 1 ]
interface [ id 2 ]
interface [ id 3 ]
interface [ id 5 ]
]
]
Net [
id 5
AS_status boundary
router [
id 1
graph [
ProtocolSession [
```

```
name bgpkiller use  SSF.OS.BGP4.Widgets.BGPKiller
kill 5000
]
ProtocolSession [
name bgp  use  SSF.OS.BGP4.BGPSession
autoconfig false
connretry_time 120  min_as_orig_time 15
ghostflushing true
reflector false
neighbor [ as 1 address 1(5) use_return_address 1(1)
_extends .basic_ebgp_neighbor ]
neighbor [ as 2 address 1(5) use_return_address 1(2)
_extends .basic_ebgp_neighbor ]
neighbor [ as 3 address 1(5) use_return_address 1(3)
_extends .basic_ebgp_neighbor ]
neighbor [ as 4 address 1(5) use_return_address 1(4)
_extends .basic_ebgp_neighbor ]
]
ProtocolSession [ name socket use SSF.OS.Socket.socketMaster ]
ProtocolSession [ name tcp    use SSF.OS.TCP.tcpSessionMaster ]
ProtocolSession [ name ip     use SSF.OS.IP ]
]
interface [ id 0  virtual true ]
interface [ id 1 ]
interface [ id 2 ]
interface [ id 3 ]
interface [ id 4 ]
]
]
link [ attach 1:1(2) attach 2:1(1) delay 1 ]
link [ attach 1:1(3) attach 3:1(1) delay 1 ]
link [ attach 1:1(4) attach 4:1(1) delay 1 ]
link [ attach 1:1(5) attach 5:1(1) delay 1 ]
link [ attach 2:1(3) attach 3:1(2) delay 1 ]
link [ attach 2:1(4) attach 4:1(2) delay 1 ]
link [ attach 2:1(5) attach 5:1(2) delay 1 ]
link [ attach 3:1(4) attach 4:1(3) delay 1 ]
link [ attach 3:1(5) attach 5:1(3) delay 1 ]
link [ attach 4:1(5) attach 5:1(4) delay 1 ]
]
```

Appendix D

SSFNet Simulation execution Makefile

```
SHELL = /bin/sh
TOPDIR = /home/gonrod-3/ssfnet
include Makefile.common
JAVAC = javac -classpath .:$(SSFNET_TEST_CLASSPATH)
JAVA  = nice -19 java  -classpath .:$(SSFNET_TEST_CLASSPATH)
JAVANOTNICE  = java  -classpath .:$(SSFNET_TEST_CLASSPATH)
SCHEMAS = $(TOPDIR)/examples/net.dml
DICTIONARY = /home/gonrod-3/ssfnet/src/SSF/OS/BGP4/test/dictionary.dml
MAXMEM = 3900

TESTNAME = withdrawals
RUNTIME = 10000

# ----- MAKE ALL ----------------------------

all:
    @echo Simulation Time: $(RUNTIME)
    @echo Max Memory: $(MAXMEM) megabytes
    @echo OutPut File: $(NOMFICH)
#   $(JAVAC) *.java
    @rm -f $(NOMFICH) test.out
    $(JAVA) -Xmx$(MAXMEM)m SSF.Net.Net $(RUNTIME) $(DMLFICH) \
        $(DICTIONARY) $(SCHEMAS) > $(NOMFICH) 2>&1
    @less $(NOMFICH) | grep -v "SYN from" > $(NOMFICH).copy
    @mv $(NOMFICH).copy $(NOMFICH)
    @echo `date`:`sh machine.sh`:Sim:$(DMLFICH) >> simulation.log
view:
    $(JAVA) RacewayViewer $(DMLFICH)
```

Appendix E
"Multi" Network Models

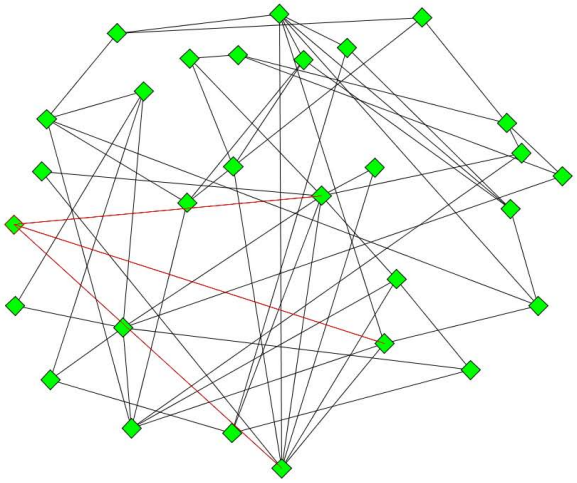

Figure E.1: "29 nodes structure'. Image Generated by RaceWay network visualization tool

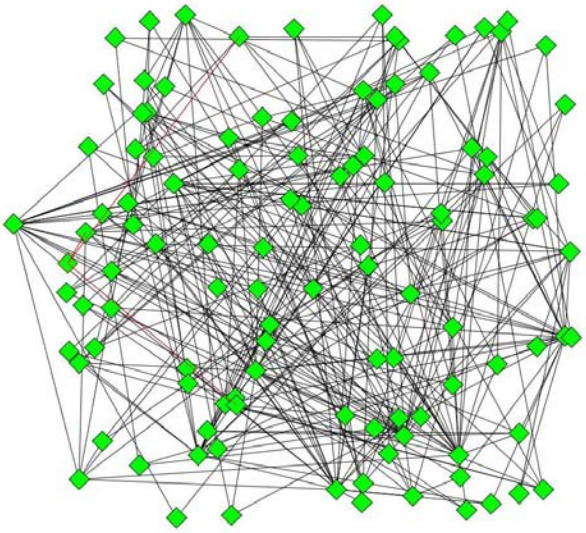

Figure E.2: "110 nodes structure'.

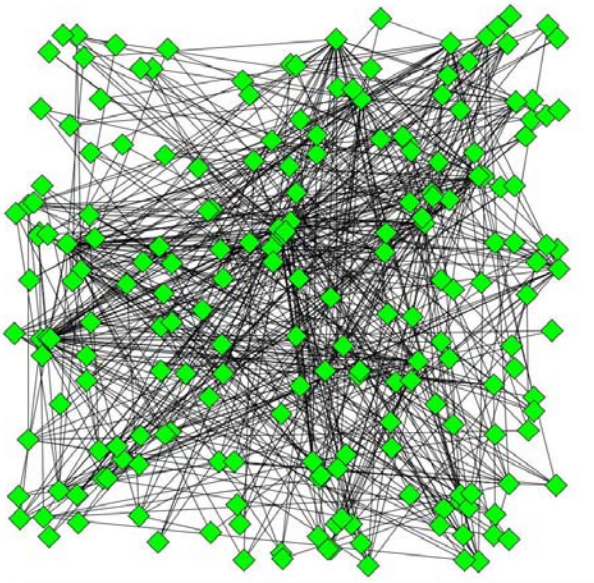

Figure E.3: “208 nodes structure’.

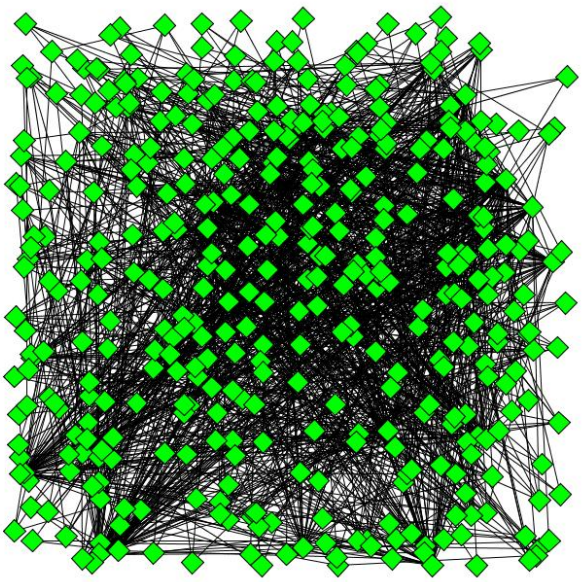

Figure E.4: “409 nodes structure’.

Appendix F
Zebra Testing Lab

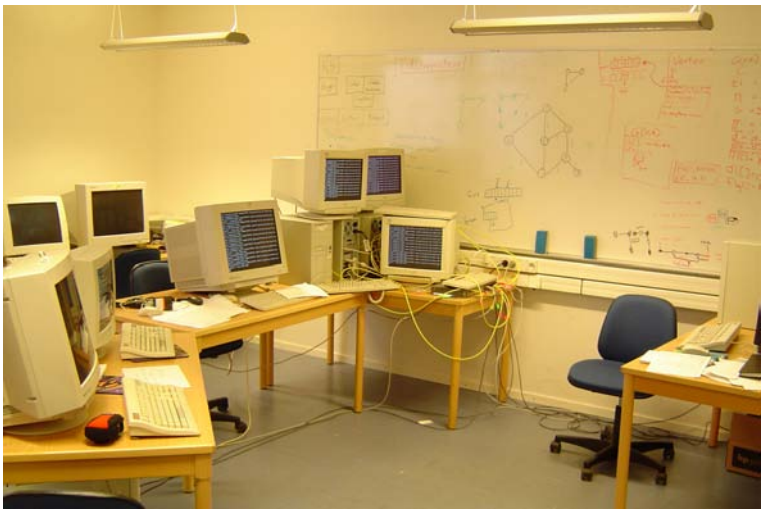

Figure F.1: Zebra testing Lab, A1203.

Figure F.2: Hp procurve switch 2524 used during the tests.

Appendix G

GNU Zebra log file example

Log file excerpt from Node 1 in a 7 nodes clique after node 7 has gone down. Ghost-flushing rule was deactivated.

```
2004/07/21 23:11:48 BGP: 192.168.0.2 send UPDATE 213.98.0.0/16 -- unreachable
2004/07/21 23:11:48 BGP: 192.168.0.5 send UPDATE 213.98.0.0/16
2004/07/21 23:11:49 BGP: 192.168.0.3 rcvd UPDATE w/ attr: nexthop 0.0.0.0, origin i
2004/07/21 23:11:49 BGP: 192.168.0.3 rcvd UPDATE about 213.98.0.0/16 -- withdrawn
2004/07/21 23:11:51 BGP: 192.168.0.6 rcvd UPDATE w/ attr: nexthop 0.0.0.0, origin i
2004/07/21 23:11:51 BGP: 192.168.0.6 rcvd UPDATE about 213.98.0.0/16 -- withdrawn
2004/07/21 23:11:52 BGP: 192.168.0.4 rcvd UPDATE w/ attr: nexthop 0.0.0.0, origin i
2004/07/21 23:11:52 BGP: 192.168.0.4 rcvd UPDATE about 213.98.0.0/16 -- withdrawn
2004/07/21 23:12:09 BGP: 192.168.0.2 rcvd UPDATE w/ attr: nexthop 192.168.0.2, origin i,
     path 2 3 7
2004/07/21 23:12:09 BGP: 192.168.0.2 rcvd 213.98.0.0/16
2004/07/21 23:12:09 BGP: 192.168.0.5 send UPDATE 213.98.0.0/16 -- unreachable
2004/07/21 23:12:09 BGP: 192.168.0.2 send UPDATE 213.98.0.0/16
2004/07/21 23:12:12 BGP: 192.168.0.3 send UPDATE 213.98.0.0/16
2004/07/21 23:12:12 BGP: 192.168.0.3 rcvd UPDATE w/ attr: nexthop 192.168.0.3, origin i,
    path 3 4 7
2004/07/21 23:12:12 BGP: 192.168.0.3 rcvd 213.98.0.0/16
2004/07/21 23:12:16 BGP: 192.168.0.4 send UPDATE 213.98.0.0/16
2004/07/21 23:12:17 BGP: 192.168.0.6 send UPDATE 213.98.0.0/16
2004/07/21 23:12:18 BGP: 192.168.0.5 rcvd UPDATE w/ attr: nexthop 192.168.0.5, origin i,
     path 5 6 7
2004/07/21 23:12:18 BGP: 192.168.0.5 rcvd 213.98.0.0/16
2004/07/21 23:12:20 BGP: 192.168.0.2 rcvd UPDATE w/ attr: nexthop 0.0.0.0, origin i
2004/07/21 23:12:20 BGP: 192.168.0.2 rcvd UPDATE about 213.98.0.0/16 -- withdrawn
2004/07/21 23:12:39 BGP: 192.168.0.2 send UPDATE 213.98.0.0/16
2004/07/21 23:12:42 BGP: 192.168.0.3 send UPDATE 213.98.0.0/16
2004/07/21 23:12:42 BGP: 192.168.0.3 rcvd UPDATE w/ attr: nexthop 192.168.0.3, origin i,
     path 3 6 4 7
2004/07/21 23:12:42 BGP: 192.168.0.3 rcvd 213.98.0.0/16
2004/07/21 23:12:46 BGP: 192.168.0.4 rcvd UPDATE w/ attr: nexthop 192.168.0.4, origin i,
     path 4 5 6 7
2004/07/21 23:12:46 BGP: 192.168.0.4 rcvd 213.98.0.0/16
2004/07/21 23:12:46 BGP: 192.168.0.4 send UPDATE 213.98.0.0/16
2004/07/21 23:12:47 BGP: 192.168.0.6 send UPDATE 213.98.0.0/16
2004/07/21 23:12:47 BGP: 192.168.0.3 rcvd UPDATE w/ attr: nexthop 0.0.0.0, origin i
2004/07/21 23:12:47 BGP: 192.168.0.3 rcvd UPDATE about 213.98.0.0/16 -- withdrawn
2004/07/21 23:12:47 BGP: 192.168.0.4 rcvd UPDATE w/ attr: nexthop 0.0.0.0, origin i
2004/07/21 23:12:47 BGP: 192.168.0.4 rcvd UPDATE about 213.98.0.0/16 -- withdrawn
2004/07/21 23:12:48 BGP: 192.168.0.5 rcvd UPDATE w/ attr: nexthop 192.168.0.5, origin i,
     path 5 2 4 1 7
2004/07/21 23:12:48 BGP: 192.168.0.5 rcvd UPDATE about 213.98.0.0/16 --
```

```
    DENIED due to: as-path contains our own AS;
2004/07/21 23:12:48 BGP: 192.168.0.2 send UPDATE 213.98.0.0/16 -- unreachable
2004/07/21 23:12:48 BGP: 192.168.0.3 send UPDATE 213.98.0.0/16 -- unreachable
2004/07/21 23:12:48 BGP: 192.168.0.4 send UPDATE 213.98.0.0/16 -- unreachable
2004/07/21 23:12:48 BGP: 192.168.0.6 send UPDATE 213.98.0.0/16 -- unreachable
2004/07/21 23:13:09 BGP: 192.168.0.2 rcvd UPDATE w/ attr: nexthop 192.168.0.2,
    origin i, path 2 3 6 4 7
2004/07/21 23:13:09 BGP: 192.168.0.2 rcvd 213.98.0.0/16
2004/07/21 23:13:12 BGP: 192.168.0.3 send UPDATE 213.98.0.0/16
2004/07/21 23:13:12 BGP: 192.168.0.3 rcvd UPDATE w/ attr: nexthop 192.168.0.3,
    origin i, path 3 5 6 4 7
2004/07/21 23:13:12 BGP: 192.168.0.3 rcvd 213.98.0.0/16
2004/07/21 23:13:14 BGP: 192.168.0.5 rcvd UPDATE w/ attr: nexthop 0.0.0.0, origin i
2004/07/21 23:13:14 BGP: 192.168.0.5 rcvd UPDATE about 213.98.0.0/16 -- withdrawn
2004/07/21 23:13:14 BGP: 192.168.0.5 Can't find the route 213.98.0.0/16
2004/07/21 23:13:14 BGP: 192.168.0.3 rcvd UPDATE w/ attr: nexthop 0.0.0.0, origin i
2004/07/21 23:13:14 BGP: 192.168.0.3 rcvd UPDATE about 213.98.0.0/16 -- withdrawn
2004/07/21 23:13:14 BGP: 192.168.0.2 rcvd UPDATE w/ attr: nexthop 0.0.0.0, origin i
2004/07/21 23:13:14 BGP: 192.168.0.2 rcvd UPDATE about 213.98.0.0/16 -- withdrawn
2004/07/21 23:13:14 BGP: 192.168.0.3 send UPDATE 213.98.0.0/16 -- unreachable
```

Appendix H
Digital Media Contents

This report attaches a CD containing the software used, modified and created in this project. Also, it includes this very same report altogether with the two presentations done for it.

So this are the contents:

./src/

- ssfnet-ghost.tar.gz
  Raceways SSFNet 2.0 Simulator with the ghost-flushing and ghost-buster modifications on the BGP4 module.
- ssfnet-sim-tools.tar.gz
  Programs developed to create, run and analyze the simulations.
- zebra-0.94-ghost.tar.gz
  GNU Zebra version 0.94, modified to use the ghost-flushing rule.
- zebra-tools.tar.gz
  Collection of tools to generate configuration files for GNU Zebra and run them.
- examplesBGP++.tar.gz
  Example and log files of the anomalous behavior of NS-2 with BGP++ as it was sent to the developers.

./report/

- proyecto-gf.pdf
  The complete report of this Master's Thesis.

./show/

- proyecto-gf-eng.ppt
  Powerpoint English presentation about this Master's Thesis.
- proyecto-gf-esp.ppt
  Powerpoint Spanish presentation about this Master's Thesis.

./RFCs/ Collection of RFCs related with this Master's Thesis

./RelatedDoc/ Articles Related with this Master's Thesis

./README.txt Data media contents description.

Bibliography

[1] *C++ Implementation of BGP over ns-2*. http://www.isi.edu/nsnam/ns/.

[2] *Connected: An Internet Encyclpedia*, volume Internet History chapter. http://www.freesoft.org/CIE/Topics/57.htm.

[3] *The GateD consortium*. http://www.gated.org.

[4] *GNU-Zebra*. http://www.zebra.org.

[5] *J-Sim*. http://www.j-sim.org.

[6] *Marc Greis' Tutorial for the UCB/LBNL/VINT Netork Simulator ns*. http://www.isi.edu/nsnam/ns/tutorial/.

[7] *The Network Simulator - ns-2*. http://www.isi.edu/nsnam/ns/.

[8] *Quagga Software Routing Suite*. http://www.quagga.net.

[9] *Scalable Simulation Framework*. http://www.ssfnet.org.

[10] *Sun Fire V480 Server Brochure*. http://www.sun.com/servers/entry/v480/.

[11] *The Wikipedia*, volume Routing Chapter. http://en.wikipedia.org/.

[12] The impact of internet policy and topology on delayed routing convergence. *ACM SIGCOMM*, 30:175–187, October 2000.

[13] S. S. Anat Bremler-Barr, Yehuda Afek. Improved bgp convergence via ghost flushing. *IEEE Infocom*, 2003.

[14] N. Bhatnagar and H. Dara. The architecture of zebra. Technical report. http://wiki.cs.uiuc.edu/cs427/Software+Architecture+of+Zebra.

[15] A. B. C. Labovitz, A. Ahuja and F. Jahanian. Delayed internet routing convergence. *ACM SIGCOMM*, August 2000.

[16] R. G. C. Villamizar, R. Chandra. Bgp route flap damping. RFC 2439, Internet Engineering Task Force, November 1998.

[17] U. P. de Madrid. *VNUML homepage*. http://jungla.dit.upm.es/~vnuml/#documentation.

[18] L. Gao. On inferring autonomous system relationships on the internet. 2000.

[19] C. Hedrick. Routing information protocol. RFC 1058, Internet Engineering Task Force, Jun 1988.

[20] C. Huitema. *Routing in the Internet*. Prentice Hall, 1995.

[21] Y. R. K. Lougheed. Border gateway protocol (bgp). RFC 1105, Internet Engineering Task Force, Jun 1989.

[22] Y. R. K. Lougheed. Border gateway protocol 3 (bgp-3). RFC 1267, Internet Engineering Task Force, Oct 1991.

[23] A. D.-S. J. Michael Feng, Roy Leung. Summer report 1999: Linux network. Technical report, Network Architecture Lab, Electrical and Computer Engineering, University of Toronto.

[24] J. Moy. Ospf specification. RFC 1131, Internet Engineering Task Force, Oct 1989.

[25] J. S. P. Traina, D. McPherson. Autonomous system confederations for bgp. RFC 3065, Internet Engineering Task Force, February 2001.

[26] B. Premore. *Multi-AS topologies from BGP routing tables.* `http://www.ssfnet.org/Exchange/gallery/asgraph/index.html`.

[27] E. R. Hinden, S. Deering. Ip version 6 addressing architecture. RFC 1884, Internet Engineering Task Force, December 1995.

[28] Y. Rekhter. A border gateway protocol 4 (bgp-4). RFC 1771, Internet Engineering Task Force, March 1995.

[29] E. Rosen. Exterior gateway protocol. RFC 827, Internet Engineering Task Force, Oct 1982.

[30] E. C. T. Bates, R. Chandra. Bgp route reflection an alternative to full mesh ibgp. RFC 2796, Internet Engineering Task Force, April 2000.

[31] P. Traina. Experience with the bgp-4 protocol. RFC 1773, Internet Engineering Task Force, March 1995.

[32] J. Y. V. Fuller, T. Li. Supernetting: an address assignment and aggregation strategy. RFC 1338, Internet Engineering Task Force, June 1992.

[33] A. R. V. Gill, D. Walton. Border gateway protocol (bgp) persistent route oscillation condition. RFC 3345, Internet Engineering Task Force, August 2002.

[34] I. van Beijnum. Running zebra on a unix machine. *O'REILLY, ONLamp.com*, 7 2002. `http://www.onlamp.com/pub/a/onlamp/2002/11/07/zebra.html`.

[35] E. Y. Rekhter, T. Li. A border gateway protocol 4 (bgp-4). RFC 1654, Internet Engineering Task Force, July 1994.

[36] yon uriarte. Zebra hacking how-to. Technical report. `http://www.quagga.net/zhh.html`.

[37] R. Zakon. Hobbes' internet timeline. RFC 2235, Internet Engineering Task Force, November 1997.